Salinas–User's Notes

Garth Reese[*] Dan Segalman[†] Manoj K. Bhardwaj[‡]
Kenneth Alvin[§] Brian Driessen Kendall Pierson[¶] Timothy Walsh[‖]

Sandia National Laboratories
Albuquerque, NM 87185-0847

April 11, 2006

[*]Phone: 845-8640
[†]Phone: 844-0972
[‡]Phone: 844-3041
[§]Phone: 844-9329
[¶]Phone: 284-5894
[‖]Phone: 284-5374

Revision: 1.48

Date: 2005/11/17 20:11:14

Latest Software Release: 2.3

Abstract

Salinas provides a massively parallel implementation of structural dynamics finite element analysis, required for high fidelity, validated models used in modal, vibration, static and shock analysis of weapons systems. This document provides a users guide to the input for Salinas. Details of input specifications for the different solution types, output options, element types and parameters are included. The appendices contain detailed examples, and instructions for running the software on parallel platforms.

(this page intentionally blank)

Salinas User Notes

Contents

List of Figures

List of Tables

Salinas

Salinas provides a massively parallel implementation of structural dynamics finite element analysis. This capability is required for high fidelity, validated models used in modal, vibration, static and shock analysis of weapons systems. General capabilities for modal, statics and transient dynamics are provided.

This document describes the input for the Salinas program. Examples of input specifications are scattered throughout the document. Appendix A provides several full input files. Appendix B provides instructions on invoking Salinas on a serial UNIX platform. Appendix C details how to execute Salinas on the ASCI-red machine, `janus`.

The name for Salinas is taken from a series of ancient Tewa Indian pueblos to the east of Albuquerque, New Mexico. These pueblos have been a source of culture and of salt for centuries. They were among the first settlements for Spanish explorers in the region.

1 Introduction – Input File

The input file contains all the directives necessary for operation of the program. These include information on the type of solution, the name of the exodus file containing the finite element data, details of the material and properties within the element blocks, which boundary conditions to apply, etc. Details of each of these sections are covered below.

Typically, the input file has an extension of "`.inp`", although any extension is permitted. If the "`.inp`" extension is used, **Salinas** may be invoked on the input without specifying the extension.

The input file is logically separated into sections. Each section begins with a keyword (**Solution**, **BLOCK**, etc), and ends with the reserved word **end**. Words within a section are separated with "white space" consisting of tabs, spaces, and linefeeds. Comments are permitted anywhere within the file, and follow the C++ convention, i.e. a comment begins with the two characters "//" and ends with the end of the line.*

Except for data within quotes, the input file is case insensitive. The software converts everything to lower case unless it is enclosed in quotes. Either the single quote `'` or the double quote `"` may be used. The quotes may be nested, e.g. `'a string with "embedded" quotes'`, but only with the other style mark.

*To be safe, define comments as "//" followed by a space.

The input parser supports nested includes. This is done using the **#include** command. This is the only command the parser recognizes. Files may be included to any depth. As an example,

```
#include english_materials
```

The **#include** may occur anywhere on the line (though for readability and consistency we recommend that it be the start of the line). The file name must immediately follow and should NOT be enclosed in quotes. Case sensitivity will be preserved. Summarizing, a minimum of two files are needed to run **Salinas** , namely, a text input file, e.g. *example.inp*, and an **Exodus** input file,[1] e.g. *example.exo*, which contains the finite element model. The finite element model is specified in *example.inp* as the geometry file (see section 2.11).

Each of the **Salinas** input sections is described in the following section.

2 The Salinas Input File

2.1 SOLUTION

The **solution** section determines which solution method, and options are to be applied to the model. The available solution types are shown in Table 1. Relevant options are shown in Table 4, and are described in section 2.2.

2.1.1 Multicase

All of the solution methods of table 1 may be a part of a multicase solution. This allows the user to specify multiple steps in a solution procedure. For example, there can be a static preload, a computation of the updated tangent stiffness matrix, and a linearized eigen analysis. The syntax for multicase solutions is similar to that for single cases, but each solution step is delineated by the "case" keyword. In addition, any of the modal solutions must be preceded by an eigen analysis and eigen keywords are no longer recognized as part of the solution.

In a multicase solution, the system matrices (mass, stiffness and damping) will typically be computed only once. Matrix updates between solutions may be specified by selecting the **tangent** keyword (see section 2.1.22).

Multicase Parameters. Many of the solution parameters are specific to a particular solution type. For example, time step parameters are meaningless in a modal solution. However, some options apply more generally. These parameters, listed in Table 2, may be specified either above the case control sections, or within the section. The specification above the case control section is the default value. Specifications within the case sub-blocks apply only to that sub-block. In the example below, the `restart` options are thus "none" for most subcases, but "read" for the eigen analysis and auto for the linear transient.[†]

Multicase Example. In the example which follows, a nonlinear statics computation is followed by a tangent stiffness matrix update. The updated matrix is then used in an eigen analysis. Two sets of exodus output files will be written. Output from the statics calculation will be in files of the form *'example-nls.exo'*. Eigen results will be in the form *'example-eig.exo'*. The **tangent** solution normally produces no output in the exodus format.

[†] These features are not yet fully implemented in release 2.0. Currently only one restart or solver option is recognized for all solutions.

Table 1: Salinas Solution Types

Solution Type	Description	Parameters
buckling	buckling eigensolution	nmodes, shift
cbr	Craig-Bampton reduction	nmodes, shift
ceigen	complex eigen	
checkout	skip large matrix and solves	
cjdamp	modal damping contributions	
continuation	nonlinear parameter continuation	LOCA
directfrf	direct frequency response	
dump	form matrices only	
eigen	real eigensolution	nmodes, shift, untilfreq
eigenk	real eigensolution of K *(seldom useful)*	nmodes
modalfrf	frequency response using modal displacement or modal acceleration	nmodes, usemodalaccel, nrbms
modalranvib	random vibration using modal superposition	eigen parameters noSVD
modalshock	shock response spectra using modal approximate implicit transient analysis (unimplemented)	nmodes, time_step, nsteps, nskip, flush srs_damp
modaltransient	transient analysis using modal superposition	nmodes, time_step, nsteps, nskip, flush
NLstatics	nonlinear statics	max_newton_iterations,tolerance num_newton_load_steps, update_tangent
NLtransient	implicit nonlinear transient analysis	time_step, nsteps, nskip, rho, flush, max_newton_iterations,tolerance
old_transient	implicit transient analysis (acceleration based)	time_step, nsteps, nskip, rho, flush *(can include sensitivity analysis)*
Receive_Sierra_Data	coupling to Sierra	
statics	static stress	
subdomain_eigen	subdomain eigenanalysis *(ONLY for debug)*	nmodes
tangent	compute tangent matrices	*(multicase only)*
transhock	shock response spectra using direct implicit transient analysis	time_step, nsteps, nskip, flush srs_damp
transient	implicit transient analysis	time_step, nsteps, nskip, rho, flush
tsr_preload	thermal structural response	file *(multicase only)*

Table 2: Multicase Parameters
These parameters may be specified as defaults above the case specifications, or they may be specified for each subcase to which they apply.

Parameter	Description	Options
restart	Restart options	see section 2.2.1
solver	selection of solver	see section 2.2.3
scale	If set to yes, turns diagonal scaling on.	see section 4
scalarstructuralacoustics	Requests a structural acoustic simulation	

```
Solution
   restart=none
   title='example multicase'
   case 'nls'
     nlstatics
     load=10
   case 'tangent'
     tangent
   case 'eig'
     eigen
     restart=read
   case 'trans1'
     transient
       restart=auto
       time_step 1e-8 1e-6
       nsteps    100  4000
       flush 50
       rho=0.9
       load=20
   case 'trans2'
     transient
       restart=auto
       time_step 1e-4
       nsteps    200
       flush 10
       load=20
END
```

The **case** keyword must always be followed by a label. The label is used in the output file name. The **case** keyword is also used to divide parameters of each

solution type.

The **load** keyword is used within a solution step to indicate which loads to apply during a solution. In the example above, load '10' will be applied during the nonlinear statics calculation. During a multicase solution the **loads** section (found elsewhere in the file) will be ignored. See paragraph 2.13 for information on the **loads** section or paragraph 2.14 for information on the **load** section of the input file.

2.1.2 A Note On Time Stepping In Multicase Solutions

In the multicase example provided above, compare cases 'trans1' and 'trans2'. It is important to note that case 'trans1' will step through 100 steps of time at a step size of 1e-8, then step through 4000 steps at a step size of 1e-6. Assuming the calculation starts at time=0, the final time value of case 'trans1' will be 1e-8*100 + 1e-6*4000 = 0.004001. Case 'trans2' will start at 0.00400099 and run an additional 200 time steps at a step size of 1e-4. This will end at a time value of 0.024001. (NOTE: This was not the default behavior for Salinas versions 1.2.1 or earlier).

2.1.3 Ceigen

The **Ceigen** keyword is used to select complex eigen analysis. This computes the solution to the quadratic eigenvalue problem,

$$\left(K + D\lambda + M\lambda^2\right) u = 0 \tag{1}$$

This capability is available in release 1.2.

The following table gives the parameters needed for complex eigen analysis.

Parameter	Argument	Default
nmodes	*Integer*	100
viscofreq	*Real*	1e-6

The **nmodes** keyword indicates the number of modes to compute in the quadratic eigenvalue analysis. These modes are computed (and reported) as complex conjugate pairs.

The optional **viscofreq** keyword indicates the frequency at which the damping properties of viscoelastic materials will be computed. It must be non-negative. The **viscofreq** parameter can be very confusing. In particular, viscoelastic materials typically have high damping at lower frequencies, and lower damping at high frequencies. The **viscofreq** parameter sets a frequency from which we estimate *all*

Table 3: Ceigen Tests

Name	**Description**
ceig	stiffness proportional damping
ceig_visco	viscoelastic damping
ceig_dash	dashpot damping
steel_in_foam	complex mixed materials

of the viscoelastic damping. Thus, if **viscofreq** is small, the damping is large. In particular, if **viscofreq** is below the glass transition frequency, then damping appropriate to the low frequency modes will be used. *This high value of damping is applied to the entire spectrum.* It is generally better to over-estimate **viscofreq** than to underestimate it.

The reason for this difficulty is that even linear viscoelastic materials generate a more complex equation than that shown in equation 1. With a single term in the Prony series, the equation of motion for a damped viscoelastic structure can be written in the frequency domain.

$$\left(K + D\frac{s}{s+\omega_g} + Ms^2\right)u = f(s) \tag{2}$$

Where s is the *Laplace* transform variable and $\omega_g = 1/\tau$ is the reciprocal of the relaxation constant. Clearly this system is not a simple quadratic in s. Effectively, **viscofreq** approximates this system with the linearized system below.

$$\left(K + D\frac{s}{2\pi \cdot \mathbf{viscofreq} + \omega_g} + Ms^2\right)u = f(s) \tag{3}$$

Computation of quadratic eigenmodes is much more difficult than real eigen analysis. The system of equations is more difficult, and more "tricks" must be used to resolve issues that are generated. Even the post processing can be complicated. Like real eigen analysis, one must request displacement output in the "output" section (see 2.8.6). Now the output file contains 12 separate fields (six real and six imaginary) for the complex results. Few post processing tools know what to do with these results. More details are provided in section 1.9 of the theory manual.

Because of the difficulties with complex eigen analysis, it is important to understand the problems for which we have evaluated and tested it. The tests in the test suite are listed in Table 2.1.3.

2.1.4 Checkout

The **checkout** solution method tests out various parts of the code without forming the system matrices or solving the system of equations. This solution method may be used to check input files for consistency and completeness on a serial platform before allocating expensive resources for a full solution.

2.1.5 CJdamp

The **CJdamp** solution provides a method of computation of the equivalent modal damping terms introduced from material damping in lightly damped visco elastic materials. It is based on a development by Conor Johnson *et al.*[2] It is an approximate method which assumes that the mode shapes and frequencies are not modified by the damping. The modal damping is simply related to the fraction of energy in block.

The **CJdamp** method is effectively a postprocessing step following an eigen analysis. For each of the modes in the eigen analysis, a strain energy is computed on an element basis. These are summed at the block level.

$$SE_j^i = \sum_{elem}^{\text{in block j}} \phi_i^T K^{elem} \phi_i \tag{4}$$

The total strain energy TSE is just the sum of the contributions from all blocks. We define the block strain energy ratio for mode i as,

$$R_j^i = SE_j^i / TSE \tag{5}$$

The **CJdamp** contribution for the modal damping of mode i, is given by,

$$\zeta_i = \frac{1}{2} \sum_j R_j^i \eta_j(f_i) \tag{6}$$

Where $eta_j(f_i)$ is the **CJetaFunction** contribution from block j evaluated at the natural frequency of mode i (see section 2.21.9).

Note that cases following the CJdamp solution will include this damping as part of their damping calculation.

Example,

```
SOLUTION
   case eig
       eigen nmodes=30
   case cjd
       cjdamp
   case frf
       modalfrf
END
```

2.1.6 Continuation

Continuation is the process of tracking a nonlinear static solution as system parameters are varied, in other words, computing the curve $u(\lambda)$ defined by

$$r(u, \lambda) = p(u, \lambda) - f(u, \lambda) = 0 \tag{7}$$

where p and f are as defined in (16). The system parameter λ may represent a component of an external applied load or moment, an element attribute, a material property, or the total external force. This last case is similar to the load stepping procedure for nonlinear static solutions defined in Section 2.1.17, and continuation can be viewed as a generalization of the load stepping procedure to a variety of system parameters.

Numerically, points along the continuation curve $u(\lambda)$ are computed using the LOCA (Library of Continuation Algorithms) library which uses the NOX nonlinear solver to compute each point on the curve. While all of the capabilities of this library are too numerous to list here (see `http://software.sandia.gov/nox` for more details), three important capabilities for structural mechanics problems are pseudo-arclength continuation, automated eigenanalysis along the continuation path, and bifurcation tracking. Pseudo-arclength continuation is a continuation algorithm that reparameterizes the continuation curve with respect to an approximate arclength, i.e., $u(\lambda) \rightarrow (u(s), \lambda(s))$. This is useful when the continuation curve as a function of the parameter λ folds over and becomes multi-valued as in snap-through or buckling. Numerically, this is implemented by adjoining the equations $r(u, \lambda) = 0$ with an additional scalar equation that constrain the Newton updates to be orthogonal to an approximate tangent to the continuation curve.

The fold-over point of the continuation curve is called a fold or turning point bifurcation, and represents a qualitative change in the dynamics of the system (for example, in snap-through or buckling, the system becomes unstable at this point). It occurs when the system Jacobian (tangent stiffness matrix) becomes singular.

LOCA has the capability to track this bifurcation point in a second parameter μ, yielding a curve in (λ, μ) space upon which the bifurcation occurs. In addition to tracking turning point bifurcations, LOCA can track pitchfork (symmetric buckling) and Hopf (onset of oscillations) bifurcations.

All three bifurcations are signaled by the real part of an eigenvalue of the system passing through zero at the bifurcation point. For turning point and pitchfork bifurcations, the eigenvalue is real near the bifurcation point, whereas for the Hopf the eigenvalue is complex with a nonzero imaginary part at the bifurcation. LOCA has the capability to compute the eigenvalues of the system at each point along the continuation curve using the Anasazi eigensolver package facilitating the location of bifurcation points.

The continuation case is selected by supplying the **continuation** keyword in the **SOLUTION** block. Any valid **NLStatics** parameter is available for continuation as well, with two important differences. The **num_newton_load_steps** keyword is not available since the continuation process implements this functionality in a more general way, and the **tolerance** is replaced by four tolerances that allow more fine-grained control of the convergence criteria. Additional parameters that can be set in the **SOLUTION** block are

Parameter	Argument	Default
residual_relative_tolerance	*Real*	1e-6
residual_absolute_tolerance	*Real*	1e-6
update_relative_tolerance	*Real*	1e-6
update_absolute_tolerance	*Real*	1e-6

Keywords **residual_relative_tolerance**, **residual_absolute_tolerance**, **update_relative_tolerance**, and **update_absolute_tolerance** supply relative and absolute tolerances controlling the convergence criteria of each nonlinear solve along the continuation curve. The nonlinear solve is considered converged if

$$\|r\|_2 < \|f_0\|_2 \varepsilon_r + \varepsilon_a \tag{8}$$

and

$$\|\Delta u\|_2 < \|u_0\|_2 \varepsilon_r + \varepsilon_a \tag{9}$$

where ε_r, ε_a are the relative and absolute tolerances, r is the current nonlinear residual, f_0 is the initial external force for that continuation step, Δu is the update to the solution components, and u_0 is the initial solution for that continuation step. Note that both of these conditions must be satisfied for convergence, not either of them as in the case of **NLStatics**.

All of the remaining options controlling the continuation and bifurcation tracking algorithms are placed in the **LOCA** block described in Section 2.31.

Note: Currently, only one continuation case is allowed in an input file. In the future, multiple cases will be supported. However, multiple continuation cases can be simulated using the restart capability described in Section 2.31. Also, the parameter **NegEigen** should be supplied in the **PARAMETERS** block (Section 2.3) to allow negative continuation parameter values to be saved in the output Exodus file (if this is not specified, Salinas will convert all negative parameter values to zero). Finally, incorporation of the system mass matrix into the Hopf tracking and eigenanalysis using Anasazi is currently not complete.

2.1.7 Craig-Bampton Reduction

It can be advantageous to reduce a model to it's interface degrees of freedom. This is very important in satellite work, where the model of the satellite may be much larger than the model of the remainder of the missile. Reduction of the satellite model to a linearized, Craig-Bampton model makes it possible to share the dynamic properties of the model without requiring details of the interior. There are many types of component mode synthesis techniques (or **CMS**), of which the Craig-Bampton approach is one of the more popular. In this approach the model is reduced to a combination of fixed interface and constraint modes. The fixed interface modes are eigenvectors of the system with all interface degrees of freedom clamped. The constraint modes are the deformations introduced when one interface degree of freedom receives a unit displacement, and all other interface degrees of freedom are zero.

The **CBR** solution reduces an entire structural model to its reduced system and transfer matrices. Parameters are listed in the table below, and correspond to the parameters required for an eigen analysis (section 2.1.10). In addition, a **CBModel** section must be defined elsewhere (see section 2.26). Any boundary conditions specified are applied before reducing the model. An example is provided below.

Parameter	Type	Argument
nmodes	*integer*	number of constraint modes
shift	*Real*	negative shift

The method will write system matrices and general information. No model file data will be written.

```
SOLUTION
  case cbr
    cbr
```

```
        nmodes=20 shift=-4e6
END
```

Note the following limitations for the CBR method.

- *In serial, no MPCs may share nodes with interface nodes. Otherwise the MPCs may eliminate the dofs that should be retained. For parallel domain decomposition solvers (such as FETI), this restriction is relaxed.*
- *Parallel solvers require a large negative shift. This is required to ensure that all subdomains are nonsingular.*
- *The entire reduced order model and associated transfer matrix must fit into memory. On a parallel machine, this memory is required on every processor. The model dimension is the sum of the number of constraint and fixed interface modes.*

2.1.8 Directfrf

Option **directfrf** is used to perform a direct frequency response analysis. In other words, we compute a solution to the Fourier transform of the equations of motion, i.e.

$$\left(K + i\omega C - \omega^2 M\right)\overline{u} = \overline{f}(\omega)$$

where $\overline{u}$ is the Fourier transform of the displacement, u, and $\overline{f}$ is the Fourier transform of the applied force. The method used is to compute the frequency dependent matrix $A(\omega) = K + i\omega C - \omega^2 M$, and frequency component of the force at each frequency point at the output. The matrix equation is then solved once per frequency point. When a direct solver is used, this means that a complex factorization must be performed once per output. This can be very time consuming, and the **modalfrf** may be a better option for many situations (see section 2.1.13).

The force function must be explicitly specified in the load section, and MUST have a "function" definition. Note that the force input provides the real part of the force at a given frequency, i.e. it is a function of frequency, not of time. At this time, we do not provide a way to input a complex force.

The parameters **freq_step**, **freq_min**, and **freq_max** are used to define the frequencies for computing the directfrf. They are identified in the **frequency** section along with an application region (see section 2.10). The range of the computed frequency response is controlled by **freq_min** and **freq_max**, while **freq_step** controls the resolution.

Note that, currently, directfrf is only a serial implementation. The parallel implementation is scheduled for implementation in release 1.2.

We note that, in addition to the output that is sent to the .frq file, output is also written to the exodus file during a directfrf, provided that the keywords are specified in the **output** section. If nothing is specified in the **output** section, then nothing is written to the exodus output files.

2.1.9 Dump

The keyword **dump** will cause **Salinas** to form matrices only and no solution will be obtained.

2.1.10 Eigen

The **eigen** keyword is needed to obtain the eigenvalues and mode shapes of a system. The parameters which can be specified for an eigensolution are shown in the table below. By default, if **nmodes** is not specified, a value of 10 is used.

Parameter	Argument	Default
nmodes	*Integer*	10
shift	*Real*	0
untilfreq	*Real*	0

The **shift** parameter indicates the shift desired in an eigenanalysis. The shift value represents a shift in the eigenvalue space (i.e. ω^2 space). The value to select is problem dependent, and only relevant for singular systems (i.e. floating structures). Please see the following discussion.

Eigenanalysis of singular systems

The eigenvalue problem is defined as,

$$(K - \omega^2 M)\phi = 0. \tag{10}$$

Where K and M are the stiffness and mass matrices respectively, and ω and ϕ are the eigen values and vectors to be determined. The problem may be solved using a variety of methods - the Lanczos algorithm is used in Salinas. In this method, a subspace is built by repeated solving equations of the form $Ku = b$. For floating structures, or structures with mechanisms, K is singular and special approaches are required to solve the system. The two approaches used in Salinas are described below.

Deflation. If it is possible to identify the singularity in K, then the null vectors of K are eigenvectors (with $\omega = 0$), and the system can be solved by insuring that no component of the null vectors ever occurs in b. This approach is equivalent to computing the pseudo inverse of K.

The strength of deflation is that if the eigenvectors can be determined exactly, the Lanczos algorithm is unaltered and the remaining vectors can be determined somewhat optimally. The difficulty is ensuring that we have correctly determined the eigenvectors, especially when mechanisms or multipoint constraints exist in the model. Determination of the eigenvectors is often a tolerance based approach that has not been as robust as we would like.

Shifting. The second method involves solution of a modified (or shifted) eigenvalue problem.

$$((K - \sigma M) - \mu M)\,\phi = 0. \tag{11}$$

This system has the properties that the eigenvectors, ϕ, are unchanged from the original equation, and the eigenvalues, μ, are simply related to the original values. Namely, $\mu = \omega^2 - \sigma$.

The shifted problem benefits from the fact that $K - \sigma M$ can be made nonsingular (except in very rare situations). This is done by choosing σ to be a large negative value. Unfortunately, the Lanczos routine convergence is affected if σ is chosen to be too far from zero[‡]. A reasonable value is $\sigma = -\omega_{elas}^2$, where ω_{elas} is the expected first nonzero (or elastic) eigenvalue.

On serial platforms we support only the shifted method. Because of the higher accuracy of direct solvers, a small negative shift is normally sufficient to solve the problem. This shift (usually -1) is computed automatically. We do not recommend that you override the defaults.

When using the FETI solver on parallel platforms both methods are available. If deflation is used, user input (and careful evaluation) may be required to ensure that all global rigid body modes have been properly identified. The relevant FETI parameters are `rbm` and `grbm_tol` as described in appendix 5.3.

The shifted eigenvalue problem has proven to be more robust for many complex problems. Set the `grbm_tol` to a small value (e.g. 1e-20), and manually enter a negative shift. The output should still be examined to insure that no global rigid body modes are detected.

[‡] If σ is too large a negative value, many solves will be required to determine the eigenvalues (which consequently slows convergence). Another consequence is that often not all redundant, zero eigenvalues may be found. They may be found by reducing the shift, tightening tolerances or by restarting.

If the model is not floating and has no mechanisms, the system is not singular, and no shift should be used (as it may slow convergence).

The **untilfreq** keyword provides an additional method of controlling the eigenspectrum to be computed. If this value is provided, then the analysis will be automatically (and internally) restarted until the frequency of the highest mode is at least the value of the **untilfreq**. This restart capability is somewhat crude. The are always **nmodes** new modes computed on each calculation. Also, because there can be inaccuracies associated with restarting the eigensolver,[§] we restart a maximum of **5** times.[¶]

Example

A **SOLUTION** section for an eigenanalysis with a **shift** of -10^6 , will look like the following, if 12 modes are needed. This shift would be appropriate for a system where the first elastic mode is approximately 150Hz.

```
Solution
  eigen
  nmodes 12
  shift -1.0e6
end
```

2.1.11 Eigenk

The **eigenk** keyword is used to obtain the eigenvalues and eigenvectors of the stiffness matrix of the model. This is equivalent to **eigen** if the mass matrix is equal to the identity matrix. The same parameters apply.
IT IS CURRENTLY ONLY AVAILABLE ON SERIAL PLATFORMS.

2.1.12 Buckling

The **buckling** keyword is used to obtain the buckling modes and eigenvalues of a system. The parameters which can be specified for a buckling solution are shown in the table below. By default, if **nmodes** is not specified, a value of 10 is used.

[§] We use the **ARPACK** Lanczos solver for the eigen problem. This solver maintains the orthogonality of the eigenvectors for a single batch of modes. However, when we restart it, we must deflate out the previously computed modes. There can thus be a slight loss of orthogonality. When we repeatedly restart, the effect can be significant.

[¶] We anticipate that in the future, this keyword will be retired when better control methods are provided.

Parameter	Argument	Default
nmodes	*Integer*	10
shift	*Real*	0

The **shift** parameter indicates the shift desired in a buckling analysis. The shift value represents a shift in the eigenvalue space (i.e. ω^2 space). The value to select is problem dependent.

The **nmodes** parameter specifies the number of requested buckling modes. Most commonly, only the critical (lowest) buckling mode is of interest, and in that case **nmodes** would be specified to be 1. However, there are cases when the first few buckling modes are of interest, and thus this parameter can be specified in the same way as in eigenanalysis.

Unlike eigenanalysis, buckling solution cases require a **loads** block. This is because buckling is always specified with respect to a particular loading configuration. For example, for a pressure load applied on a sideset, the buckling analysis would indicate the critical amplitude of the applied pressure needed to cause buckling. The critical buckling load is computed as the product of the first (lowest) eigenvalue times the amplitude of the applied load. Thus, for the case

```
LOADS
  sideset 1
  pressure = 10.0
END
```

and a lowest obtained eigenvalue of 100.0, the critical buckling pressure would be computed as $P_{cr} = 100.0 \times 10.0 = 1000.0$. This would indicate that buckling would occur if the loading were applied as,

```
LOADS
  sideset 1
  pressure = 1000.0
END
```

Similar conclusions can be drawn about force loads on nodesets.

Buckling solutions cannot be computed for floating structures. If there are global rigid body modes, the solution may not be correct. Also, for meshes with MPCs, only parallel solution is possible. Serial buckling solutions with MPCs cause a fatal error in the constraint transformations. This error will be eliminated in future versions.

One additional constraint on buckling is that currently beams and shells cannot be used in buckling solutions. We expect to eliminate this restriction in future releases.

Example

A **SOLUTION** section for buckling analysis with a **shift** of -10^6 , will look like the following, if only 1 mode is needed (i.e. if only the critical buckling load is of interest).

```
Solution
  buckling
  nmodes 1
  shift -1.0e6
end
```

2.1.13 Modalfrf

Option **modalfrf** is used to perform a modal superposition-based frequency response analysis. In other words, the modalfrf provides an approximate solution to the Fourier transform of the equations of motion, i.e.

$$\left(K + i\omega C - \omega^2 M\right) \overline{u} = \overline{f}(\omega)$$

where $\overline{u}$ is the Fourier transform of the displacement, u, and $\overline{f}$ is the Fourier transform of the applied force.

Two options are available for the modalfrf solution: the modal displacement method, and the modal acceleration method. In both cases the approximate solution is found by linear modal superposition. Once the modes have been computed, there is little cost in computation of the frequency response. The solution does suffer from modal truncation of course, but in the case of the modal acceleration method a static correction term partially accounts for the truncated high frequency terms. Thus, in general that method is more accurate than the modal displacement method. The most accurate, but also the most computationally expensive approach is the **directfrf** method described in section 2.1.8.

For the modal displacement method, the relation used for modal frequency response is given below.

$$\overline{u}_k(\omega) = \sum_j \frac{\phi_{jk}\phi_{jm}\overline{f}_m(\omega)}{\omega_j^2 - \omega^2 + 2i\gamma_j\omega_j\omega}$$

Here $\overline{u}_k$ is the Fourier component of displacement at degree of freedom k, ϕ_{jk} is the eigenvector of mode i at dof k, and ω_j and γ_j represent the eigenfrequency and associated fractional modal damping respectively.

For the modal acceleration method, the procedure for computing the modal frequency response is more complicated. The response is split into the rigid body contributions, and the flexible contributions. The number of global rigid body modes must be specified in the input file. For details on the theory, we refer to section 1.8 of the theory manual.

The modal acceleration method is typically much more accurate at finding the zeros of a function, but only slightly more accurate in finding the poles (or peaks) of the response. The cost is an additional factor and solve. It can be used on floating structures, but the additional factor involves only the stiffness terms (which are singular) and has no mass terms to stabilize the solution. Thus, it may be much more difficult to perform that solve than the other solves involved in the eigen analysis. In eigen analysis we recommend a negative shift for floating structures to remove the singularity associated with rigid body modes. No such approach is possible if you are using the modal acceleration method. Thus, significant "tweaking" of the FETI parameters may be required to accurately determine the global rigid body modes required for success of this method.

The force function must be explicitly specified in the load section, and MUST have a "function" definition. Note that the force input provides the real part of the force at a given frequency, i.e. it is a function of frequency, not of time. At this time, we do not provide a way to input the imaginary component of the force.

The following table gives the parameters needed for **modalfrf** section.

Parameter	Argument
nmodes	*Integer*
usemodalaccel	-
nrbms	*Integer*

The **nmodes** parameter controls the eigenanalysis (see section 2.1.10). The optional keyword, **usemodalaccel**, is used to determine whether to use the modal displacement or the modal acceleration method. If this keyword is specified, modal acceleration is used, otherwise the modal displacement method is invoked. If **usemodalaccel** is used, then the number of global rigid body modes must be specified using **nrbms**.

The parameters **freq_step**, **freq_min**, and **freq_max** are used to define the frequencies for computing the shock response spectra. They are identified in the

frequency section along with an application region (see section 2.10). The range of the computed frequency spectra is controlled by **freq_min** and **freq_max**, while **freq_step** controls the resolution. The accuracy of the computed spectra is not dependent on the magnitude of **freq_step**. This parameter only controls the quantity of output.

We note that, in addition to the output that is sent to the .frq file, output is also written to the exodus file during a modalfrf, provided that the keywords are specified in the **output** section. If nothing is specified in the **output** section, then nothing is written to the exodus output files.

2.1.14 Modalranvib

Option **modalranvib** is used to perform a modal superposition-based random vibration analysis in the frequency domain. The solution computes the root mean square (RMS) outputs (including the von mises stress) for a given input random force function. The resulting power spectral density functions may also be output for locations specified in the "frequency" section. The forcing functions (one for each input) must be explicitly specified in the load section, and MUST have a "matrix-function" definition (see section 2.24).

The following table gives the **solution** parameters needed for **modalranvib** analysis.

Parameter	Argument
nmodes	*Integer*
noSVD	N/A
lfcutoff	*Real*
keepmodes	*Integer*

The **nmodes** parameter controls the eigenanalysis (see section 2.1.10). All keywords associated with eigen analysis are appropriate and available. It is recommended that the eigenanalysis be performed as the first step of a multicase solution.

The *optional* keyword **noSVD** determines the method used to compute the RMS von Mises stress output. If **noSVD** is specified, then the simpler method which does not use a singular value decomposition is used. However, this method provides no information about the statistics of the stress. Only the RMS value is reported.

The *optional* keyword **lfcutoff** provides a low frequency cutoff for random vibration processing. Usually, rigid body modes are *not* included in this type of calculation. The **lfcutoff** provides a frequency below which the modes are ignored. The default for this value is 0.1 Hz. Thus, by default rigid body modes are not

included in random vibration analysis. A large negative value will include all the modes.

The *optional* keyword **keepmodes** is a method of truncating modes. By default, its value is **nmodes**. If a value is provided, the modes with the lowest modal activity will be truncated until only **keepmodes** remain. Note that this is a much different truncation procedure than simply truncating the higher frequency modes. Modal truncation is important because all of the operations compute responses that require order N^2 operations. Even if **keepmodes** is not entered, modes with modal activity less than 1 millionth of the highest active mode will be truncated.

The parameters **freq_step**, **freq_min**, and **freq_max** are used to define the frequencies for computing the random vibration spectra. They are identified in the **frequency** section along with an optional application region (see section 2.10). The range of the computed frequency spectra is controlled by **freq_min** and **freq_max**, while **freq_step** controls the resolution. The accuracy of the computed spectra *does* depend on the magnitude of **freq_step** since it is used in the frequency domain integration.

In random vibration, the **frequency** block serves two purposes. First, it is used for the integration information for the entire model. Thus Γ_{qq} for the referenced papers[3] is integrated over frequency and used for all output. In addition, if an output region is specified in the **frequency** block, output acceleration and displacement power spectra may be computed for the given region at the required frequency points. At this time, only "acceleration" and/or "displacement" may be specified in the frequency block for random vibration analysis. This output is described in more detail below.

Random vibration analysis is a little trickier than most input. A number of blocks must be specified.

1. The **solution** block must have the required input for eigen analysis, and the keyword **modalranvib**.

2. The **RanLoads** block contains a definition of the spectral loading input matrix and the loadings. Note that the input, S_{FF} is separated into frequency and spatial components. The spatial component is specified here using **load** keywords. See section 2.15. The spectral component is referred to here, but details are provided in the matrix-function section.

3. The **matrix-function** section contains the spectral information on the loading. It references functions for the details of the load. The real and imaginary function identifiers for this input are specified here (2.24).

4. There must be a **function** definition for each referenced spectral function. Functions of time or frequency are further described in section 2.23.

5. There must be a **frequency** block that is used for integration and optionally also for output of displacement and acceleration output. See section 2.10.

6. As an undamped system is singular, some type of **Damping** block information needs to be provided. Modal damping terms are required. See section 2.28.

7. **Boundary** conditions are supplied in the usual way, but the standard **loads** block is replaced by the input in the ranloads section. The loads block will be quietly ignored in random vibration analysis.

8. The **output** and **echo** sections will require the keyword **vrms** for output of RMS von mises stress. If the *stress* keyword is also found, then the natural stresses for solid elements will be output.†

All other input should remain unchanged.

Power Spectral Densities. One output from the random vibration analysis is a power spectral density or PSD (for displacement or acceleration). The power spectral density is a measure of the output content over a frequency band, and usually measured in units of cm^2/Hz or some similar unit. Acceleration PSDs are often measured in units of g^2/Hz.‡

Like the input cross spectral forces, the output quantities are hermitian, with 9 independent quantities at each frequency, at each output node for each type of output. Details of how these quantities are transformed in alternate coordinate systems are outlined in section 1.8 of the theory manual. The matrix quantities are diagrammed below. Quantities are output in the order A_{xx}, A_{yy}, A_{zz}, A_{zx}, A_{zy}, A_{xy}, A_{zxi}, A_{zyi}, A_{xyi}.

$$\begin{bmatrix} A_{xx} & A_{xy}+iA_{xyi} & A_{xz}+iA_{xzi} \\ A_{xy}-iA_{xyi} & A_{yy} & A_{yz}+iA_{yzi} \\ A_{xz}-iA_{xzi} & A_{yz}-iA_{yzi} & A_{zz} \end{bmatrix}$$

Because the inputs are specified in terms of force cross-correlation functions, the standard procedure for applying loads often involves application of a large concentrated mass at the input location. The force may then be applied to the mass and the acceleration determined from $a = f/m$, where we assume that m is much larger

† The natural stresses are output in the following order: σ_{xx}, σ_{yy}, σ_{zz}, σ_{yz}, σ_{xz}, σ_{xy}. These stresses are linear functions of the displacement.

‡ Power spectral density output is requested in the **frequency** block. A collection of nodes is indicated and the *displacement* or *acceleration* keyword is entered. PSDs of displacement or acceleration are available.

than the mass of the remainder of the structure. Some confusion can arise in the scaling of the force.

The output PSD for acceleration is defined as follows.

$$G_{ij} = H^{\dagger}_{ki} S_{kl} H_{lj} \tag{12}$$

$$< a_i a_j > = H^{\dagger}_{ki} < f_k f_l > H_{lj} \tag{13}$$

where H_{lj} is the transfer function giving a_j/f_l.
Consider a single input, i.e. $k = l$, and with $f_k = m_k a_k$.

$$G_{ij} = H^{\dagger}_{ki} < m_k a_k a_k m_k > H_{lj} \tag{14}$$

$$= (m_k^2) H_{ki} < a_k a_k > H_{kj} \tag{15}$$

Thus, the acceleration PSD must be multiplied by the square of the mass to get the force PSD. Note that **Salinas** uses the scale factor in the spatial force distribution, so the scale factor in **Salinas** should be m_k.

2.1.15 Modalshock

The **modalshock** solution method is used to perform a modal superposition-based implicit transient analysis followed by computation of the shock response spectra for the degrees of freedom in a specified node set. The following table gives the parameters needed for **modalshock**.

Parameter	Argument
nmodes	*Integer*
time_step	*Real*
nsteps	*Integer*
nskip	*Integer*
srs_damp	*Real*

The **nmodes** parameter controls the modal solution described in section 2.1.10. The time stepping parameters **time_step**, **nsteps** and **nskip** are described in the transient section (2.1.24).

The parameters **freq_step**, **freq_min**, and **freq_max** are used to define the frequencies for computing the shock response spectra. They are identified in the **frequency** section along with an application region (see section 2.10). The range of the computed frequency spectra is controlled by **freq_min** and **freq_max**, while

freq_step controls the resolution. The accuracy of the computed spectra is not dependent on the magnitude of **freq_step**. This parameter only controls the quantity of output.

The optional parameter **srs_damp** is a damping constant used for the shock response spectra calculation. Its default value is 0.03. Damping for the model is defined in section 2.28.

2.1.16 Modaltransient

Option **modaltransient** is used to perform a modal superposition-based implicit transient analysis. The following table gives the parameters needed for **modaltransient**. Damping for the model is defined in section 2.28.

Parameter	Argument	default
time_step	*Real*	none
nsteps	*Integer*	none
nskip	*Integer*	1
load	*Integer*	*sec 2.14*

The parameters **time_step**, which defines the time integration step size, and **nsteps**, which defines the total number of integration steps, are required. The optional parameter **nskip** controls how many integration steps to take between outputting results. (It defaults to 1, which is equivalent to outputting all time steps). Time dependent loadings are applied by referencing the appropriate **load** and **function** sections (see 2.14 and 2.23).

Modal transient should normally be executed as a later step of a multicase solution, where previous steps computed the eigenvalue response. However, for compatibility with earlier formats, **modaltransient** can be called as a single step solution (see section 2.1.10). In that case the following eigen value parameters are also required. Note that in a single step solution (with no case structure), no **load** keyword is required, but a **loads** section must exist in the file (see section 2.13).

Parameter	Argument
nmodes	*Integer*
shift	*Real*

The parallel solution of modal transient may be slower than expected because while the eigen solution parallelizes very well, there is not enough computation to parallelize the modal calculation. In addition, Salinas computes the displacements at all locations in the model before subsetting to those nodes in the history file.
If output is only required at a few locations, you may want to consider a matlab integration. You will need the eigenvalues and vectors from the history file, and the modal generalized forces. These forces are written to `'ModalFv.m'`.

2.1.17 NLStatics

The **NLstatics** keyword is required if a nonlinear static solution is needed, i.e. the solution to the system of equations $[K]\{u\} = \{f\}$, where K is now a function of u. The following table gives the parameters needed for nonlinear static analysis.

Parameter	Argument	Default
max_newton_iterations	*Integer*	100
tolerance	*Real*	1e-6
num_newton_load_steps	*Integer*	1
update_tangent	*Integer*	101

Four parameters control the conventional Newton method. Newton methods are nonlinear solution algorithms employed to solve the residual force equations. The residual vector, r, is the difference between the internal force vector, p, and the external force vector, f. The strategy drives the residual to zero.

$$r = p - f \tag{16}$$

The internal force vector is a function of the structural displacements (and possibly velocities). External forces can also be a function of the structural displacements in the case of follower loads such as surface pressure loads. [§]

The **tolerance** provides control over the completion of the newton iteration. Once the change in the L2 norm of *displacement* decreases below **tolerance**, the loop completes successfully. If the iteration count exceeds **max_newton_iterations**, the Newton loop is considered to have failed.

The **num_newton_load_steps** keyword controls the number of load steps used to incrementally step up to the final equilibrium position. Large loads may

[§]Follower loads are not currently supported.

cause the Newton algorithm to diverge. If this occurs, increase the number of load steps applied. Displacements will be output after each load step which may be animated similar to transient dynamics simulations.

The **update_tangent** keyword controls how often the tangent stiffness matrix is rebuilt during the Newton iterations. The default is set to update the tangent stiffness matrix at the beginning of a load step only. Setting **update_tangent** to 1 is equivalent to using a full-Newton algorithm where the tangent stiffness matrix is rebuilt after each Newton iteration. For highly nonlinear (difficult) problems, this option may be optimal, but for most problems the extra cost incurred in recomputation and refactoring of the tangent stiffness matrix should be amortized over several solves. Note, for this option to improve Newtons method, the element types in the model have to have the tangent stiffness method implemented.

An example **SOLUTION** section is shown below.

```
Solution
  title 'Example of a nonlinear statics solution'
  nlstatics
  tolerance             = 1e-6
  max_newton_iterations = 100
  num_newton_load_steps = 10  // split load into 10 increments
  update_tangent        = 1   // full-newton algorithm
end
```

2.1.18 NLTransient

The **NLtransient** solution method is used to perform a direct implicit nonlinear transient analysis. The following table gives the parameters needed for nonlinear transient analysis.

The nonlinear transient analysis is performed according to methods described in Hughes. A projector, corrector step is used. Note that for a linear system the NLtransient analysis will require two solves per time step.

Parameter	Argument	Default
time_step	*Real*	-
nsteps	*Integer*	-
nskip	*Integer*	1
flush	*Integer*	50
rho	*Real*	Newmark beta
max_newton_iterations	*Integer*	100
tolerance	*Real*	1e-6
num_newton_load_steps	*Integer*	1
update_tangent	*Integer*	101

The time step control parameters, **time_step**, **nsteps**, **nskip** and **flush** are described in the **transient** section above, section 2.1.24. The parameter **rho** is the same as described in the previous section. We note that, as in the case of linear transient analysis, multiple time steps can be specified in nonlinear transient analysis. The syntax for this is the same as described in the section on linear transient analysis.

Four parameters control the conventional Newton method used to solve the residual force equations. The **tolerance** provides control over the completion of the newton iteration. Once the change in the L2 norm of *acceleration* decreases below **tolerance**, the loop completes successfully. If the iteration count exceeds **max_newton_iterations**, the Newton loop is considered to have failed.

The **num_newton_load_steps** controls the number of load steps used to incrementally step up to the final equilibrium position. Large loads may cause the Newton algorithm to diverge. For nonlinear statics, it is recommended to increase the number of load steps. For nonlinear transient problems, if Newtons method diverges, either the tangent stiffness matrix has to be updated more often (see **update_tangent**) or the time-step should be decreased. The default value is 1 for both nonlinear statics and nonlinear transient solution methods.

The **update_tangent** controls how often the dynamic tangent stiffness matrix is rebuilt during the Newton iterations. The default is set to update the dynamic tangent stiffness matrix at the beginning of a load step. Setting **update_tangent** to 1 is equivalent to using a full-Newton algorithm where the dynamic tangent stiffness matrix is rebuilt after each Newton iteration. For highly nonlinear problems, some control of this option is recommended. Note, for this option to improve Newtons method, the element types in the model have to have the dynamic tangent stiffness method implemented.

2.1.19 Receive_Sierra_Data

Coupling of Salinas Through The Sierra Framework

Calculations in Sierra codes such as **Presto**, may be transferred to Salinas. This provides the ability to compute very nonlinear responses in an explicit code, and follow that by an implicit, mildly nonlinear calculation in Salinas.

A solution method named **Receive_Sierra_Data** facilitates the transfer of data, which may occur either through the sierra framework or an Exodus input file written from the sierra application.

The method used for the transfer depends on the executable built. As currently configured, the standard salinas executable must use the file transfer. A specially linked executable, **Prelinas**, is used for the direct memory transfer of data.

Salinas was coupled with Presto in version 2.0, and additional couplings are under development. Further information is found in the *Tempo* documentation. The Receive_Sierra_Data solution method may be used with sierra enabled executable, or with the sierra input file (see section 2.11.3).

The "Receive_Sierra_Data" solution makes sense only in the context of a multi-case solution. There are no parameters. An example follows.

```
SOLUTION
    case xfer
       receive_sierra_data
    case eig
       eigen nmodes=40 shift=-3e6
END
```

2.1.20 Statics

The **statics** keyword is required if a static solution is needed, i.e. the solution to the system of equations $[K]\{u\} = \{f\}$. An example **SOLUTION** section is shown below.

```
Solution
   title 'Example of a statics solution'
   statics
end
```

2.1.21 Subdomain_Eigen

The **subdomain_eigen** keyword is used to obtain the eigenvalues and eigenvectors of the mass and stiffness matrix of the model on a subdomain basis. This is useful mainly for debugging distributed solutions. It is obviously decomposition dependent, and has no physical meaning. The parameters are listed below.

Parameter	Argument	Default
nmodes	*Integer*	10
shift	*Real*	0

Many domain decomposition tools (such as FETI-DP) depend on non-singular subdomain stiffness matrices. Running subdomain_eigen on these systems reveals the condition of the system that is to be solved. For FETI-DP, the system of interest is the subdomain defined with the corner nodes clamped. This can be determined using the following procedure.

1. Set the FETI parameter **prt_debug**=3 in the FETI section (see section 2.4.2). Running a standard analysis (i.e. statics, transient analysis or eigen) will output the "`corners.data`" file. This file should normally be written properly even if the analysis fails.

2. Copy the file to a new name, and modify it to contain only the global node ids. This is the first column of the file.

3. Use the **node_list_file** option to clamp the corner nodes in the file (see section 2.12.3).

4. Run salinas using the **subdomain_eigen** option. Ask for 14 modes or so. A very small first mode indicates a singular system for which our corner selection algorithm has not properly constrained the subdomain.

2.1.22 Tangent

The **tangent** solution step is only relevant as part of a multicase solution (see paragraph 2.1.1). It forces an update of the tangent stiffness matrix. It is typically used following a nonlinear solution step to insure that the following step begins using the tangent stiffness matrices computed from the previous result. However, it may also be used following a linear solution step, in which case the stiffness matrix is recomputed based on the current value of displacement.

The tangent stiffness matrix is assembled at the subdomain level from computations at the element level. It represents the partial derivative of the force with respect to the displacement, i.e.

$$K_{tangent} = \frac{\partial f}{\partial u} \tag{17}$$

In eigen analysis, the tangent stiffness matrix replaces the linear stiffness matrix in the eigenvalue equation. This permits computation of modal response following a preload. In nonlinear transient dynamics, the tangent stiffness matrix is used in the Newton (or other) iteration scheme used to reduce force residuals.

2.1.23 Transhock

The **transhock** solution method is used to perform a direct implicit transient analysis followed by computation of the shock response spectra for the degrees of freedom in a specified node set (all node sets are defined in the Exodus file). The following table gives the parameters needed for transient shock analysis.

Parameter	Argument
time_step	*Real*
nsteps	*Integer*
nskip	*Integer*
srs_damp	*Real*

The parameters **time_step**, which defines the time integration step size, and **nsteps**, which defines the total number of integration steps, are required. The parameter **nskip** controls how many integration steps to take between outputting results and is optional. (It defaults to 1, which is equivalent to outputting all time steps).

The parameters **freq_step**, **freq_min**, and **freq_max** are used to define the frequencies for computing the shock response spectra. They are identified in the **frequency** section along with an application region (see section 2.10). The range of the computed frequency spectra is controlled by **freq_min** and **freq_max**, while **freq_step** controls the resolution. The accuracy of the computed spectra is not dependent on the magnitude of **freq_step**. This parameter only controls the quantity of output.

The keyword **srs_damp** is a damping constant used for the shock response spectra calculation and is optional. It represents the damping for each single degree of freedom oscillator in the shock spectra computation. Its default value is 0.03.

Note: Currently, the shock spectrum procedure will only compute acceleration results. The options specified in the OUTPUT and ECHO blocks are used in the

transient portion of the analysis, but are ignored for the post-processing of the transient results into shock spectra. Thus, if displacement, velocity, and/or acceleration is selected in the OUTPUT and/or ECHO sections for a shock spectra analysis, the results echoed to the output listing or the Exodus output file will be time history results as requested, but the only shock spectra results will be for acceleration response for the nodes in the specified node set. Furthermore, *the calculated shock spectra will only be echoed to the output listing; they are not output to the Exodus results file.* The shock spectra output options will be revised and improved in future releases.

2.1.24 Transient

The **transient** solution method is used to perform a direct implicit transient analysis. The following table gives the parameters needed for transient analysis.[†]

Parameter	Argument	Default	Purpose
time_step	*Real*	1	set the time step
nsteps	*Integer*	100	set the number of steps
nskip	*Integer*	1	set output frequency
flush	*Integer*	50	control file buffering
rho	*Real*	none - see below	select time integrator

The parameters **time_step**, which defines the time integration step size, and **nsteps**, which defines the total number of integration steps, are required. The parameter **nskip** controls how many integration steps to take between outputting results and is optional. (It defaults to 1, which is equivalent to outputting all time steps).

The parameter **flush** controls how often the **Exodus** output file buffers should be flushed. Flushing the output insures that all the data that has written to the file buffers is also written to the disk. This parameter also controls the frequency of output of restart information if requested. Too frequent buffer flushes can affect performance. However, in a transient run, data integrity on the disk can only be assured if the buffers are flushed. A **flush** value of -1 will not flush the **Exodus** output file buffer until the run completes. The default value is to flush the buffers every 50 time steps.

We note that multiple time step values, along with the corresponding number of steps, can be specified for transient analysis. This can be useful for separating

[†] In addition to the displacement based linear transient dynamics driver, there is an older, acceleration based driver. The old driver may be selected using the **old_transient** keyword. This driver is not recommended unless sensitivity analysis is required. It is no longer fully maintained, and will be removed in future releases.

the simulation into a section of small time steps followed by a section of larger time steps, or vice versa. The following provides an example of the use of multiple time steps.

```
solution
  time_step     1e-5     1e-3
  nsteps        100      500
  nskip         10       1
end
```

In this case, the user requested 100 time steps of $\Delta t = 1E-5$, followed by 500 time steps of $\Delta t = 1E-3$. There is no practical limit on the number of such regions that may be specified.

Integrator selection

Two time integrator schemes are available for direct time integration. The method and the parameters of the integrator are selected using the keyword **rho**. If this keyword is not found, the time integrator defaults to a standard Newmark-Beta integration scheme[‡]. If the **rho** parameter is used, then the generalized alpha method[45] is used, and the value of the numerical damping is controlled by **rho**.

**** IMPORTANT ****

Because of limited accuracy in the solvers, the Newmark-Beta integrator is conditionally unstable. If no damping is provided, it occasionally diverges as time progresses. This is described in a little more detail in section 1.1 of the theory manual. Therefore it is strongly recommended that either proportional damping or numerical damping be used in all time integration.

The parameter **rho** defines the Numerical damping of the generalized alpha method. **Rho** varies from 0 (maximal damping case) to 1 (minimal damping case). **If rho is not specified in the input file, the integrator defaults to the Newmark beta method.** Otherwise, the code uses the value of **rho** given by the user to compute the parameters needed for the generalized alpha method. Therefore, there is no value default for **rho**, as shown in the table above, since if it is not specified the code uses the Newmark beta method instead. If **rho** is specified to be greater than 1 or less than 0 an error message is printed. The three parameters

[‡]The Newmark-Beta integration is described in detail in most finite element text such as Cook or Hughes.

newmark_beta, α_f, and α_m in the generalized alpha method are computed automatically, given the value of **rho**, and thus these need not be specified by the user. More detailed information on the implementation, and references can be found in the description of the method in the Salinas program_notes and theory manual.

In order to achieve second order accuracy and unconditional stability, we must satisfy the following conditions.

$$\begin{aligned} \alpha_m < \alpha_f <= \frac{1}{2} \\ \gamma_n = \frac{1}{2} - \alpha_m + \alpha_f \\ \beta_n \geq \frac{1}{4} + \frac{1}{2}(\alpha_f - \alpha_m) \end{aligned} \tag{18}$$

The code automatically computes these parameters such that they meet these criteria. Specifically,

$$\begin{aligned} \alpha_f &= \rho/(1+\rho) \\ \alpha_m &= (2\rho - 1)/(1+\rho) \\ \beta_{\text{Newmark}} &= (1 - \alpha_m + \alpha_f) \cdot (1 - \alpha_m + \alpha_f)/4 \\ \gamma_{\text{Newmark}} &= 1/2 - \alpha_m + \alpha_f \end{aligned}$$

Unlike the proportional damping parameters, there is no direct relation between **rho** and an equivalent modal damping term. Numerical damping strongly affects only the highest frequency modes (which are non-physical anyway). We recommend a value of `rho=0.9` for most analyses.

2.1.25 TSR_Preload

The **tsr_preload** solution method reads an exodus file with a previously computed Thermal Structural Response (TSR) into Salinas for a subsequent statics or transient dynamics analysis. This is not a fully coupled calculation. Rather, stress results are read from the file, an equivalent internal force is computed, and that internal force is combined with the applied force throughout the transient run. A **tsr_preload** may only be specified as part of a multicase solution, and it must be followed by a transient dynamics solution (see paragraphs 2.1.1 and 2.1.24 respectively).

Note that since the stresses are actually converted into a force, and since there is no immediate deformation in transient dynamics, the elastic stresses output by Salinas will be very small initially, i.e. they will not contain a contribution from the

thermal stress. However, at large times, the deformation from the internal force will result in an elastic stress opposite to that of the thermal stress. There is currently no method of recovering the input thermal stress as an output quantity.

The **tsr_preload** solution method is considered to be a temporary solution to a more complicated problem. In the future, TSR analysis will involve coupling to other mechanics codes.

The following table gives the **solution** optional parameter used in **tsr_preload** analysis.

Parameter	Type	Argument
file	*string*	exodus_file_name

The `exodus_file_name` is a string that points to the file containing the stress results from the TSR calculation. If no file keyword is provided, the data is expected in the input genesis file, i.e. the **geometry_file** specified in the **FILE** section (see paragraph 2.11). Currently, for parallel execution, the data must be specified in the genesis file, as the file name is not properly parsed for spread files.

Data in the exodus file must strictly match these criteria. There must be only one time step in the result. That time step must have a number of different element fields defined. These correspond to the six stresses and up to 27 different integration points of a hex20. Other solid elements are also supported. For those elements only the number of integration points applicable to that element are used. Unused integration values will be ignored. If in doubt, provide the extra integration data as missing integration points do NOT provide an error - rather they set the value to zero. Shell and beam type elements are not supported in tsr_preload.

The labels for the stresses must be as shown in the table below. In each case, replace %d with an integer representing the integration point value (0 to 26).

Name	Definition
`SIGXX_%d`	σ_{xx}, the xx component of stress
`SIGYY_%d`	σ_{yy}, the yy component of stress
`SIGZZ_%d`	σ_{zz}, the zz component of stress
`SIGYZ_%d`	σ_{yz}, the yz component of stress
`SIGXZ_%d`	σ_{xz}, the xz component of stress
`SIGXY_%d`	σ_{xy}, the xy component of stress

The **linedata_only** keyword indicates that no system matrices should be computed, but the linedata specified in the `linesample` file should be computed (see section 2.11.2). This is for verification of data transfer.

The following is an example solution section for a TSR preload followed by

transient dynamics.

```
SOLUTION
   title 'Pure bending from initial stress'
   case tsr
        tsr_preload
        load 1
   case 'trn'
      transient
      time_step 1.e-6
        nsteps 3
        nskip 1
      load 2
END
```

If executed on a file with `geometry_file='example.exo'`, this will produce two output files, `example-tsr.exo` and `example-trn.exo`. The first of these has very little useful information. The second will contain the displacements (or other variables) from the transient analysis.

One additional feature that has been added for thermal structural response is the ability to do line sampling on the original exodus file containing the element stresses. This can be useful for debugging and verification. It allows the stresses along lines within the structure to be examined.

As described in the section on FILES, 2.11, by including the name of the text file containing the line sampling information, the line sampling capability is invoked. The format of this text file is as in the following example

```
10
-1 -1 -1 1 1 1
0 0 -1 0 0 1
-1 0 0 1 0 0
```

In this case, there are 10 samples per line, as designated by the first line. The next three lines in this file denote the coordinates of the begin and end points of the lines that are to be sampled. Thus, in this example there are three lines that are to be sampled, and along each line the stresses will be sampled at 10 points. An output file, currently named linedata.m, will be generated that contains the six components of stress at all of the points along the lines. In addition, the coordinates of the sampling points are generated for purposes of visualizing the stresses.

2.2 Solution Options

The options described in Table 4 and in the following paragraphs are part of the **Solution** section in the input file. None of the keywords are required. Note that in multicase solutions most of these parameters may be applied separately within the subcase (see section 2.1.1).

Table 4: Salinas Solution Options

Option	Description	Parameters
restart	restart options	**none**, read, write or auto
lumped	Use lumped mass matrices	none
solver	Identify solver used	"auto"
constraintmethod	method of applying MPCs	Lagrange or Transform
scale	toggles diagonal scaling	"yes" or "**no**"
scalarstructuralacoustics	Request structural acoustics	

2.2.1 ReStart – option

Option **restart** controls restart file processing. Restart files permit the solution to be saved for later use. Only a limited capability is provided, but it is intended to meet most of the typical needs for structural dynamics. Note that the restart files are independent of the exodus output, but the restart options may significantly affect the exodus outputs. Application of restarts in specific sections is detailed in the following paragraphs.

There are four values associated with this option.

none indicates that restart files will be ignored. They will be neither read, nor written. Existing restart files will not be altered in any way. Restart=none is the default selection if no restart options are entered in the solution block.

read indicates that existing restart files will be read, but no output restart files will be written. If the restart files do not exist, a fatal error will result.

write indicates that existing restart files will be ignored, but restart files will be written.

auto is a combination of read and write. However, unlike read, the existence of previous restart files is optional, i.e. there will be no error message if there are no existing restart files. Invalid restart files will produce a warning, but not a fatal error.

Restarts are designed to insure accuracy of the solution. However, restarts in Salinas are not transparent in the sense that there will be small differences in two solutions to a problem when one solution involves a restart. Restarts may also have an expense. For example, the FETI solver uses an acceleration technique where the values of previous solutions are used as a starting place for new solves. The information associated with previous solutions is not stored in the file.

For transient dynamics, the state of the machine at the most recent time step is recorded. To avoid problems with corruption of a database, the three vectors (disp, velocity, acceleration) are recorded at each time step, but on alternate locations in the file. If previous exodus files exist, they will be appended. Data is written at the same interval as the exodus output.

When restarting a multicase solution, the current time is used to determine which case the restart will begin. For example, assume the following solution block is defined.

```
Solution
  case one
    transient
    restart=auto
    time_step 1e-6
    nsteps 200
  case two
    transient
    restart=auto
    time_step 1e-5
    nsteps 300
End
```

If restarting at TIME=1E-4, case "one" has a final time value of T_F = T_0 + 200*1E-6 = 2E-4, assuming T_0=0. Since TIME < T_F, case 'one' will restart

the solution. If restarting at TIME=2E-5, then case 'one' will not perform any calculations. Case 'two' will then be tested to see if a restart will begin there.

Restart Solution Support. Restarts are not supported in all solutions types. They are supported for the following.

- Eigen
- transient
- nltransient
- modaltransient

Note that none of the modal solutions except modaltransient support restart. Typically most of the computation time for these solutions is in the eigen analysis.

2.2.2 NOX

The **NOX** solution option allows the nonlinear solution procedure for either NLstatics or NLtransient problems to be driven by the NOX nonlinear solver library. Currently, Salinas can be built to run with NOX on most platforms. For such builds, the NOX solver is not used by default but can be turned on by including the **NOX** keyword as follows.

An example **SOLUTION** section is shown below.

```
Solution
  title 'Example of a nonlinear statics solution'
  nlstatics
  tolerance             = 1e-6
  max_newton_iterations = 100
  num_newton_load_steps = 10  // split load into 10 increments
  update_tangent        = 1   // full-newton algorithm
  nox   // use NOX nonlinear solver
end
```

This option has no effect other than to invoke additional parsing of the input deck for problems other than NLstatics and NLtransient. Simply using NOX in this way causes the default solution procedure to be used. This corresponds to the Newton-based approach that is used in Salinas as described in sections 2.1.17 and 2.1.18). Other solution strategies provided by the NOX library are described in section 2.30.

It should be noted that the NOX solver interface to Salinas is new and will likely contain some bugs. If found, please e-mail a description of the bug to Russell Hooper at rhoope@sandia.gov. At this time, there is a known bug with parallel execution for problems involving element types of dimension two or three. This is currently being addressed. In addition, support for other platforms is also anticipated to be provided shortly.

2.2.3 Solver

As Salinas evolves, various solvers are available for computation of the solution. Each solver brings with it different capabilities and sometimes unwanted features. Currently available solvers are listed in the following.

AUTO Use the best known solver. Generally this is recommended. The matrix of solvers versus solution types is messy, and generally the best solution will be found by using this option. For example, there is no need to change the solver as you move from serial to parallel solutions.

CLOP Under development by Clark Dohrmann, this is a domain decomposition solver that is similar to FETI in many ways.

CLIP Under development by Clark Dohrmann, this is a multigrid solver.

FETI-DP This solver is the workhorse for parallel solutions. A full description of the solver is beyond the scope of this users manual (references are on the web). FETI-DP was developed by Charbel Farhat, Kendall Pierson and others.[6] It is very scalable, and robust. Multipoint constraints are handled using Lagrange multipliers. The parallel solution process must be used with the solver, but it can be reduced to a single subdomain. Care must be used to insure that subdomains are mechanism free.

CF_FETI An evolution of FETI-DP, this solver adds the capability to compute nonlinear constraints within the solver. This is an advanced method of computing gap and contact response. It is a development platform, and currently requires a separate executable, i.e. you cannot have this solver and FETI-DP in the same executable. The CF (Charbel Farhat) solver is templated software that supports complex solves as well as real. Thus it can be used for direct frf calculations. The *CF* solver is provided for general availability on most platforms in release 2.0. Further details are available in section E.

Software limitations restrict building the CF solver with other FETI solvers. Access to this solver is through a separate executable (named *salinas_cf*).

Genfac This is the only solver currently available in serial solutions. It is a direct solver, and is part of sparspak developed by Esmond Ng. The solver is fairly robust, but may fail for singular systems. It occasionally has problems for very small systems. Originally written as a Cholesky decomposition, it has been extended to compute LDL^T. Constraints are eliminated using a transformation matrix method.

Prometheus Developed by Mark Adams, this is another multigrid solver. It has some very nice features such as augmented Lagrange constraint handling. NOT CURRENTLY AVAILABLE.

SuperLU This package, available from NERSC, provides both serial real and complex solutions. In salinas, the complex version is used for solution of serial direct FRFs.

NOX This package, currently being developed at Sandia and available at
http://software.sandia.gov,
can be used in conjunction with any of the available linear solvers to drive the nonlinear solution procedure for NLstatics and NLtransient problems. It may not be specified as "solver=nox" since another linear solver must be used. See section 2.2.2 for more details about NOX.

Generally no user input is required for specification of a solver. Indeed, up to version 1.0.5 of Salinas, only one solver was ever available at any time (i.e. we built separate executables if another solver was desired). Usually the specification can be left off, or specified as "auto". If a solver is requested and unavailable, a warning will be issued, and "auto" will be selected.

The solver may be specified as a default (above the `case` keywords as detailed in section 2.1.1), or it may be individually specified within the case framework. The default value is "auto". In the example shown below FETI-DP will be used for the eigen analysis, FETI-DPC for transient dynamics, and the "auto" selection for the direct frequency response. If "input" is specified in the "echo" section (see section 2.7) then the solver information will be echoed to the results file.

```
SOLUTION
  solver=auto
  case eig
    eigen nmodes=50
    solver=feti-dp
  case nlt
    nltransient
```

```
    solver=clop
    (other parameters)
  case frf
    directfrf
END
```

2.2.4 Lumped – option

Option **lumped** in the **SOLUTION** section causes**Salinas** to use a lumped mass matrix, and not a consistent mass matrix, in the analysis.

2.2.5 Constraintmethod – option

The **constraintmethod** option is defined in the **SOLUTION** section to indicate how multipoint constraints (MPC) will be applied. The selections for applying MPCs are are **Lagrange** and **Transform**. These methods are explained in detail on pp. 272-278 in Ref. 7.

The **constraintmethod** is currently superfluous. When using the `FETI` solver, a Lagrange multiplier method is the only method available. When using the serial solvers, the only available method is **Transform**.

2.2.6 Scale – option

Option **scale** can be set to yes or no, and controls whether diagonal scaling is applied to the system matrices before handing them over to the linear solver. In most cases, diagonal scaling improves the condition number of the local and global linear systems, and thus reduces the number of iterations required for convergence. The default behavior is not to do diagonal scaling, but when the option **scale** is set to yes, the scaling is turned on. Scaling is currently only implemented for linear and nonlinear transient dynamics. For real eigen analysis, scaling is not available. For complex eigen analysis, scaling is done internally in Salinas.

2.2.7 scalarstructuralacoustics – option

Option **scalarstructuralacoustics** is used to request a structural acoustic simulation. In order to run a structural acoustic simulation, the keyword 'scalarstructuralacoustics' must appear in the SOLUTION block. This alerts the code to assemble the coupling matrices that couple the fluid/solid responses.

The current structural acoustic capability in Salinas has the following requirements and restrictions.

- The fluid and solid meshes must be conformal along the wet interface. This implies a single mesh in which the nodes along the wet interface coincide and thus have four degrees of freedom per node (three structural displacement, one velocity potential). For parallel simulations, the structural acoustic mesh can be decomposed in the same way as with standard structural meshes.

- Both interior and exterior problems can be simulated.

- Currently, transient and directfrf structural acoustic simulations are possible. Complex eigenanalysis is expected to be in place at some time in the future.

2.3 PARAMETERS

This *optional* section provides a way to input parameters that are independent of the solution method or solver. Only one **parameter** section is recognized in each file. The parameters and their meanings are listed below.

WtMass This variable multiplies all mass and density on the input, and divides out the results on the output. It is provided primarily for the english system of units where the natural units of mass are actually units of force. For example, the density of steel is $0.283 lbs/in^2$, but "lbs" includes the units of g, 386.4 in/s^2. Using a value of wtmass of 0.00259 (1/386.4), density can be entered as 0.283, the outputs will be in pounds, but the calculations will be performed using the correct mass units.

Salinas, like most finite element codes, does not manage the *units* of the analysis. The selection of a consistent set of units is left to the analyst. For example, if the analyst uses the SI system (Kg,m,s) the units of pressure must be Pascals. Frequencies are reported in Hz. For micromachines these units are quite awkward. It may be better to use units of grams, millimeters and microseconds. The analyst must insure that all material properties and loads are converted to these units.

Some examples of useful units are shown in Table 5.

NegEigen Unconstrained structures have zero energy modes which may evaluate to small negative numbers due to machine round off. The eigenvalues and associated eigenfrequencies are reported as negative numbers in the results files. However, many post processing tools (such as *ensight*) require non-negative frequencies. By default, Salinas converts all negative eigenvalues to

Table 5: Some useful combinations of units.

length	mass	time	wtmass	density	force	modulus	intrnl mass
m	Kg	sec	1	Kg/m^3	N	N/m^2 or Pa	Kg
ft	slug	sec	1	slug/ft^3	lbf	lb/ft^2	slug
ft	lbm	sec	1/32.2	lbm/ft^3	lbf	lb/ft^2	slug
in	lbm	sec	1/386.4	lbm/in^3	lbf	psi	lbm/g
mm	μg	μs	1	Kg/m^3	N	MN/m^2 or MPa	μg
mm	g	sec	1	g/mm^3	μN	N/m^2 or Pa	gram
mm	mg	sec	1/1000	g/cm^3	μN	N/m^2 or Pa	gram

near zero values in the output exodus files[§]. To retain the negative eigenvalues in the output file, select parameter **NegEigen**.

OldBeam This option is provided for backwards compatibility with older beam models. Early Patran models using the exodus preference numbered the attributes incorrectly. The first versions of Salinas used that numbering. With the new numbering the code had to change. Providing "oldbeam" in the parameters section selects the old numbering. The new numbering will be used by default. At some point in the future, we plan to eliminate this option.

Table 6: Beam Attribute Ordering

Attribute	1	2	3	4	5	6	7
old numbering	area	orientation			I1	I2	J
new numbering	area	I1	I2	J	orientation		

eig_tol This is the tolerance used by **ARPACK** for eigensolution. If not provided, an automatic value is used.

MaxResidual This is a tolerance used to check the rigid body mode vectors calculated by `FETI`. If this residual on the rigid body mode vector is larger than this tolerance, **Salinas** will abort. The default value is 1.0.

[§]Because many postprocessing tools are written for transient dynamics, they expect monotonically increasing, positive values for the time. Since eigenvalues are written in the time columns of the output file, they are converted to be monotonically increasing, positive values. Note that the numerically computed eigen frequencies are also stored as global variables in the file

LinkStiffness This option makes it easier for some solvers to properly compute the response when there are many rigid links. At present, only RBARS and RRODS (see sections 3.31 and 3.30) are affected. The option causes **Salinas** to compute additional stiffness terms that would be associated with a beam (or truss) in place of the rigid element. Since the constraint limits the deformation to zero, there is no affect on the final solution, but the solution process can be significantly simplified since singularities are removed from the stiffness matrix. Specify `LinkStiffness=yes` or `LinkStiffness=no`. The default value is `yes`, which means the additional stiffness terms are used.

nonlinear_default In nonlinear transient dynamics or nonlinear statics, computing the fully nonlinear response of all of the elements in the mesh may be very expensive, and in some cases it is not necessary to do so. For example, for a simulation that only involves Joint2G elements and solid (3D) elements, the analyst may determine that the nonlinear effects of the solid elements are negligible. In such cases, it is advantageous to be able to control the nonlinear response of elements on a block-by-block basis. In section 2.19 of this manual, a block-level parameter is described that turns the nonlinearities on and off for individual blocks. In order to avoid having to enter this parameter for each block, the **nonlinear_default** keyword allows the user to set the default for all blocks. If it is set to no, then all blocks default to linear behavior (unless specified otherwise in the BLOCK section), and if it is set to yes, then all elements default to nonlinear behavior. Note that the block-level flags override the **nonlinear_default** keyword. There are two possible cases for this keyword.

nonlinear_default=no All elements default to linear behavior.

nonlinear_default=yes All elements default to nonlinear behavior.

TangentMethod The tangent stiffness matrix may be used in a full Newton update in nonlinear statics or transient dynamics (see sections 2.1.17 and 2.1.18). By default, each of the elements can compute it's own tangent stiffness matrix. There are cases (particularly when elements are under development) when it is better to use a tangent matrix computed from finite difference methods. There are three possible values for this keyword.

TangentMethod=element The standard element method.

TangentMethod=difference Use finite difference.

TangentMethod=compare Use the standard method, but also compute the matrix by the difference method. Unless "none" is specified in the

ECHO section (2.7), output of the difference of every element matrix in the model will be sent to the results file.¶

Info Salinas outputs many different details to standard out. Most of the details are for the developers. Many such things output are number of processors, and time taken in certain loops. Also in some cases, the contents of an array or other such storage type are output to the screen.

In many cases, this information is not wanted. The "info" option controls the output to standard out. There are four different levels of control. Each level increasingly allows more output to standard out. However, currently only two levels are supported. The other two levels of control will added in the future.

The FETI block option "prt_debug" overrides "info" when it comes to FETI output. In all other cases, "info" takes precedence. If there is no "prt_debug" command in the FETI block, then FETI output levels are also determined by "info."

The four levels of control are:

0. **Silent** – Will only output warnings and std error to the screen
1. **Normal** – Will only output the kind of data most analysts would use
2. **Detailed** – Not currently implemented. Convergence, solution addressing issues.
3. **Debug** – All of the above, plus output deemed important for debugging.

Example of usage:

```
Parameters
  info=0
End
```

This sets the "info" control level to Silent.

SkipMpcTouch Salinas uses a unique method of determining an active degree of freedom set. Unlike codes like Nastran which use an auto-spc method, Salinas loops through all elements and activates only degrees of freedom that are required for elements. Multipoint constraints pose a particular problem because some codes (like Nastran) may include multipoint constraints to unused degrees of freedom. Since these are eliminated with the autospc, this poses no

¶ In parallel solutions the results file is written only for the first processor unless the "subdomains" option is specified in the echo section (2.7).

problem to these codes, but may confuse salinas significantly. On the other hand, usually degrees of freedom associated with mpcs should be included in the active set, and leaving them out can produce errors.

As a stopgap measure, we provide the parameter **SkipMpcTouch**. If this parameter is set, no degrees of freedom will be activated through multipoint constraints.

condition_limit Element quality checks are important for evaluating the effectiveness of the mesh. By default elements with moderately bad topology are reported. However, sometimes there are so many of these warnings, that the really bad elements may get missed. The **condition_limit** parameter permits user control of the reporting. Setting this parameter to a larger number will eliminate message from marginal elements. Element checking can also be turned off (see the *elemqualchecks* parameter in the **output** section 2.8.5). The default value is 1.0.

Example,

```
Parameters
  WtMass=0.001
  Eig_Tol=1e-6
  MAXRESIDUAL=1e-3
  tangentmethod=element
End
```

reorder_rbar This option allows **RBAR**s to be reordered so that the number of **RBAR**s connected to a single node is minimized. Having a large number connected to the same node results in a highly populated matrix and a slow computation. Therefore, reducing the number of connections can shorten runtime.

If redundant **RBAR**s are present (i.e. connections forming a cycle), they are removed.

Specify `reorder_rbar yes` or `reorder_rbar no`. The default value is `yes`, which means **RBAR**s will be reordered.

thermal_time_step For thermal analysis solution procedures (i.e. statics or transient dynamics with a **thermal_load** body load), or for any solution procedure that uses temperature dependent material properties, the temperature distribution of the structure must be read in from the exodus file. Typically,

the input exodus files in this case would be the output files from a Calore analysis, and thus would contain the necessary temperature data. Since such an analysis could contain several time steps of temperature data, the parameter **thermal_time_step** allows the analyst to select which set of temperature data is to be read into Salinas. The following gives an example.

```
PARAMETERS
  thermal_time_step 10
END
```

In this case the user would be requesting that the temperature data corresponding to the 10^{th} time step be read into Salinas.

2.4 FETI

This *optional* section provides a way to input parameters specific to the Finite Element Tearing and Interconnecting[6] (FETI) solver, if used. If the FETI solver is not used, this section is ignored. It includes the following parameters, shown in Table 7, and options. For those options which are strings, only enough of the string to identify the value is required. The defaults are shown in the following example.

```
FETI
      rbm  geometric
      scaling no
      preconditioner dirichlet
      max_iter 400
      solver_tol 1.0e-5
      orthog 1000
      rbm_tol_svd 1.0e-10
      rbm_tol_mech 1.0e-8
      projector standard          // ignored in dp
      level 1                     // ignored in dp
      local_solver sparse
      coarse_solver sparse
      grbm_tol 1e-6
      prt_summary yes
      prt_rbm yes
      prt_debug 3
      corner_dimensionality 6     // for dp only
      corner_algorithm 3          // for dp only
```

Table 7: FETI Section Options

Variable	Values	Description
rbm	Algebraic/Geometric	rigid body mode method
scaling	Yes/No	scaling method
preconditioner	LUMped/DIRichlet	(both may be used)
max_iter	*Integer*	maximum number iterations
solver_tol	*Real*	
stag_tol	*Real*	Used to detect stagnation
orthog	*Integer*	max number of orthog. vectors
rbm_tol_svd	*Real*	SVD tolerance in rigid body modes
rbm_tol_mech	*Real*	mechanical tolerance in rbm
projector	**Standard**/Q	projector
level	1	feti1 (feti2 not implemented)
corner_dimensionality	*Integer*	3 or **6** dofs/corner
corner_algorithm	*Integer*	1, 3, 5-8
corner_augmentation	*String*	**"none"**, "subdomain", "edge"
local_solver	AUto, SKyline, SParse, serial_sparse	solver for local LU decomp
precondition_solver	Same as local_solver	solver for preconditioner Only used if using Dirichlet preconditioner
coarse_solver	AUto, SKyline, SParse PSparse, serial_skyline, serial_sparse	solver for coarse G^TG problem (psparse is parallel sparse)
grbm_tol	*Real*	tolerance for rigid body detection in G^TG
prt_summary	Yes/**No**	print summary timer information
prt_rbm	Yes/**No**	print # rbm in each subdomain
prt_debug	integer	debug output. values **0=none**, 1-3
bailout		if set, the solver will continue even if the solution is not converged at each intermediate solve
mpc_method	*Integer*	**0**=Lagrange multipliers everywhere 1=Local elimination where possible

```
        corner_augmentation none     // for dp only
END
```

2.4.1 Corner Algorithms

Corner selection is an important issue (and an ongoing research area) for **FETI-DP**. Several algorithms are available. They all vary by the total number of corners picked in the model for the coarse problem. The various algorithms are intended to give a little more power to the advanced user. The more corners that are picked, the quicker the solution will converge. The disadvantage being that there might not be enough memory available for these corners, hence, **Salinas** might abort because of this memory depletion. Memory statistics can be observed and with experience, the advanced user can pick the optimal corner algorithm. The possible choices for the various parameters are given in Table 8. All the options for each corner parameter are listed such that the first option for each parameter picks the least amount of corners.

Typically, corner algorithm 15 selects the minimal number of corner points. This is a useful option to try if memory becomes an issue when running on large numbers of processors. As noted above, smaller coarse grids increase the number of iterations to convergence.

Corner algorithm 14 selects three corners between along the interface between two neighboring subdomains (Γ_{ij} designates the interface between subdomain i and subdomain j). The first node is selected as the node along Γ_{ij} that touches the most subdomains. The second node is the node that maximizes the distance between any two nodes along Γ_{ij}. The third node is selected to maximize the triangular area created by three non-colinear nodes along Γ_{ij}. Corner algorithm 14 will typically select less corner nodes than Corner algorithm 3.

> *Note that additional corner nodes can be placed in a special file,* `extraNodes.dat`*. Nodes in this file will be added to the current corner selection algorithm. While this method is seldom useful, it can help in cases where an isolated element is causing catastrophic problems. The format of* `extraNodes.dat` *is to simply put the* **global** *node numbers, one per line, in the file.*

2.4.2 Levels of Diagnostic Output

This section is under construction.

Table 8: Corner Options

Parameter	**Option**	**Description**
Algorithm	0	Picks 1 corner per interface
Algorithm	1	Most robust algorithm
Algorithm	2	Picks 2 corners per interface
Algorithm	3	Picks 3 corners per interface
Algorithm	9	Picks all interface nodes *debug only*
Algorithm	14	Improved version of Corner Algorithm 3
Algorithm	15	Improved version of Corner Algorithm 0
Algorithm	16	No automatic corners. (uses `extraNodes.dat`).
Algorithm	17	like 3, but add corners for conms
Algorithm	99	like 14, but add will not pick corners on mpcs
Dimensionality	3	Fixes 3 translational d.o.f. per corner
Dimensionality	6	Fixes *all* d.o.f. per corner
Augmentation	none	no additional corners are selected
Augmentation	edge	Additional corners on interface edges are selected. (Stiffness weighted).
Augmentation	subdomain	Additional corners per subdomain are selected.

The **prt_debug** flag takes various values from 0-4. Table 9 shows the various values and their result. Note, for **prt_debug** value of 3, a file named *corner.data* is written. The format is as follows:

```
Ncorners
GlobalId LocalId SubdomainId Xpos Ypos Zpos
.
.
.
GlobalId LocalId SubdomainId Xpos Ypos Zpos
```

Ncorners is the total number of corners, *GlobalId* is the global id of the corner, followed by the local id (*LocalId*), the subdomain on which the corner exists (*SubdomainId*), and the coordinates of the corner (*Xpos Ypos Zpos*).

Table 9: Prt_Debug Options

prt_debug value	Result
0	No Output
1	Some Output
2	Lot of Output
3	Output + Corner.data file
4	Output + Corner.data file + matlab files

2.5 Clop

The "clop" solver may be specified in the solver section (see section 2.2.3). Parameters for the **Clop** solver can be specified in an *optional* "clop" section.[‖] Parameters are listed in table 10. An example follows.

```
CLOP
  max_iter=1000
  solver_tol=1e-5
  orthog=200
  prt_summary=1
  prt_debug=0
  overlap=1
END
```

[‖] *Note that the "clop" section only specifies the linear solver parameters. The "solver=clop" specification is required in the solution section.*

Table 10: CLOP Section Options

Variable	**Values**	**Dflt**	**Description**
bailout	*keyword*		If keyword is found, we try to complete the solve even if errors are found.
coarse_solver	*string*	direct	either *direct* or *3level*
krylov_method	*integer*	0	0=PCG, 1=gmres, 2=pcg (no dot product check), 7 (for structural acoustics)
local_solver	*integer*	1	1=direct
max_iter	*integer*	400	maximum number of iterations
num_rigid_mode	*integer*		number of rigid body modes for mode acceleration approach in modal transient analysis
orthog	*integer*	200	number of orthogonalization vectors *don't set this above 200.*
overlap	*integer*	0	number of overlapping elements
prt_summary	*integer*	0	output flag: 0 - no summary 1 - basic pass/fail diagnostics 2 - more yet
prt_debug	*integer*	0	0 - no debug output. 1 - too much
scale_option			0 no scaling, 1 use scaling in factorizations
solver_tol	*real*	1e-6	relative convergence tolerance
stag_tol	*real*	1e-6	stagnation tolerance

Comments:

The "orthog" option is very memory intensive right now. Do not set this to a value above 200 for the time being. Krylov_method 7 should be used for structural acoustics. It uses gmres, but with a different preconditioner that scales the acoustic and structural unknowns. This assures that both variables will be solved to the same tolerance.

2.6 CLIP

The "clip" solver may be specified in the solver section (see section 2.2.3). Parameters for the **Clip** solver can be specified in an *optional* "clip" section. Parameters are identical to those for the CLOP solver of section 2.5, and are listed in table 10.

2.7 ECHO

Results, in ASCII format, from the various intermediate calculations may be output to a results file, e.g. *example.rslt*, where the filename is generated by taking the basename of the text input file (without the extension) and adding *.rslt* as an extension. Output to the results file is selected in the **Salinas** input file using the **ECHO** section. An example is given below, and the interpretation of these keywords is shown in Table 11.

```
echo
    materials
    elements
    jacobian
    all_jacobians
    timing
    mesh
    echo
    input
    nodes
    mpc
end
```

Note that if **none** is used, the order of selection is important. Thus, if you add **none** at the end of the list, no output will be provided in the echo file. However, if you put **none nodes** then only nodal summary information will be included. Entering **nodes none mesh** only outputs the mesh information (**nodes** information is canceled by the **none**).

Table 11: ECHO Section Options

Option	Description
acceleration	nodal accelerations (better in output section)
all_jacobians	jacobians for every element
block	block wise mass properties (used only following mass)
displacement	nodal displacements (better in output section)
echo	dumb echo of input *(for parse errors)*
eforce	element force for beams
elements	element block info, i.e. what material, element type, etc
ElemEigChecks	element eigenvalue ratios
energy	element strain energy and strain energy density
eorient	element orientation
feti_input	
force	applied forces (better in output section)
genergies	global kinetic and strain energy sums
input	summaries of many sections
jacobian	block summary of jacobians
kdiag	full diagonal of stiffness matrix
mass	mass properties in the basic coordinate system
materials	material property info, e.g. E, G
mesh	summary of data from the input **Exodus** file
mesh_error	mesh discretization error metrics
NLresiduals	turns on residual output per iteration of the Newton loop for non-linear solution methods
nodes	nodal summary
pressure	applied pressures (better in output section)
rhs	Right Hand Side vector (better in output section)
subdomains "0:3:6,10"	Controls which processor will output results file
timing	timing information
velocity	nodal velocities (better in output section)
residuals	residual vectors
resid_only	residual vectors at each iteration (no other output permitted)
strain	element strains at centroids
stress	element stresses at centroids
vonmises	von mises stress only
vmrs	RMS quantities (random vibration only)
mpc	mpc equations
all	everything
none	nothing

2.7.1 Mass Properties

The mass properties may only be reported in this section (i.e. at this time there is no mass property report in the **outputs** section). The mass properties reports the total mass, the center of gravity and the moments of inertia of the system. All are reported in the basic coordinate system. Note that moments are about the origins, not about the center of gravity. Masses are reported in a unit system consistent with the input, whether or not the **WtMass** parameter has been used (see section 2.3).

An additional option of **block** may be used in the echo section to output the block wise mass properties to the results file. Please note that the block wise mass properties, though summed for all processors (if running on a parallel machine), are only output to the result file from the first processor (processor 0). The block wise mass properties option, called **block**, reports the number of blocks, the mass of each block, and the center of gravity of each block along the x, y, and z axis. Please note that **block** may only be used in the **ECHO** section just following the **mass** option like so:

```
echo                OR     echo                OR     echo
    materials                  materials                  materials
    elements                   elements                   elements
    mass=block                 mass block                 mass
    nodes                      nodes                      block
end                        end                            nodes
                                                      end
```

The prefered method is **mass=block**. However any of the previous examples will work, so long as there is whitespace between **mass** and **block**. If **mass** does not directly precede **block** in the **ECHO** section, then Salinas will give the following error:

```
Unrecognized "echo" option 'block".
Aborting.
```

2.7.2 Mpc

The keyword **mpc** instructs the code to write out the mpc equations to the result file. This is a good tool for debugging purposes, as well as a check on the input deck. An example of the output format is as follows

```
MPC
  coordinate 0
    25 P 1
   106 P -1
  // G = 0.000000
  // the source is global
END
```

In this case, the mpc equation is constraining the acoustic pressure in nodes 25 and 106 to be equal in the global (default) coordinate system.

In parallel calculations, one results file is written per subdomain. Only data associated with that subdomain are written to the file. Use the "subdomains" option to specify which subdomains for which data will be written. See the comments in the paragraphs below.

2.8 OUTPUTS

The **outputs** section determines which data will be written to selected output files. All geometry based finite element results are written to an output exodus file. The name of this file is generated by taking the base name of the input exodus geometry file, and inserting *-out* before the file extension. For example, if the input exodus file specification is *example.exo*, output will be written to *example-out.exo*. When using a multicase solution *(section 2.1.1)*, the case identifier is used in place of "out". More details are available in the **FILE** section (2.11).

Various non-geometry based finite element data, such as system matrices and tables may be available in Matlab compatible format, or in Harwell-Boeing format. These ASCII files have the *.m* or *.hb* file extensions respectively. The base file names are derived from the type of data being output. These files are generated in the current working directory.

In the following example, the mass and stiffness matrices will be output in Matlab format, but the displacement variables, stresses and strains will not be output. All the various options of the **OUTPUT** section are shown in Table 15. The next sections describe each of the options and their results assuming an input file named *example.inp* and a geometry file named *exampleg.exo*.

```
OUTPUTS
        maa
        kaa
        faa
//      displacement
```

```
//      stress
//      strain
//      energy
END
```

2.8.1 Maa

Option **maa** in the **OUTPUTS** section will output the analysis-set mass matrix to a file named *example_Maa.m*. If the harwellboeing option is selected, output will also go a file named *example_Maa.hb*.

2.8.2 Kaa

Option **kaa** in the **OUTPUTS** section will output the analysis-set stiffness matrix to a file named *example_Kaa.m*. If the harwellboeing option is selected, output will also go a file named *example_Kaa.hb*.

2.8.3 Faa

Option **faa** in the **OUTPUTS** section will output the analysis-set force vector to a file named *example_Faa.m*. If the harwellboeing option is selected, output will also go a file named *example_Faa.hb*.

2.8.4 ElemEigChecks

Option **ElemEigChecks** will turn on the element output of the first flexible eigenvalue, the largest eigenvalue, and the ratio of the two. The output will be stored in the Exodus output file. The element variable names for the smallest flexible eigenvalue, largest eigenvalue, and ratio of the two are *elam_min*, *elam_max*, and *elam_rat*, respectively. Note: All 3-d and 2-d elements have this capability. The **Beam2**, **OBeam**, **Spring**, **Truss**, **Spring3**, and **RSpring** elements are also supported. All remaining elements will output values of zero. Finally, if *elam_max/elam_min* is greater than 10^{20}, then the value of *elam_rat* will be set to 1e20.

2.8.5 Elemqualchecks

Option **Elemqualchecks** takes either one of three choices, **on**, **off**, or **sum**. The default is **sum**. If this option is **on** or **sum**, then all of the elements in the input file are checked for quality using methods developed by Knupp (Ref. 8). Knupp uses a

condition number to evaluate the health of an element. The following table shows the elements currently checked and their acceptable ranges. The element quality reporting may also be modified by the *condition_limit* parameter specified in the *Parameters* section (2.3).

Element Type	Full Range	Acceptable Range
Hex8	$1 - \infty$	$1 - 8$
Tet4	$1 - \infty$	$1 - 3$
Tria3	$1 - \infty$	$1 - 1.3$
TriaShell	$1 - \infty$	$1 - 1.3$
Quad4	$1 - \infty$	$1 - 4$
Wedge6	$1 - \infty$	$1 - 5$

If the option **on** is selected and the element's condition numbers falls outside the acceptable range, a warning message is printed. The value output with the warning is normalized by the maximum number of the acceptable range for that element. If the option **sum** is selected, only a summary is printed, reporting the maximum condition number of all elements in the mesh.

In addition to these checks, solid elements are checked for negative volumes. This can occur if the node ordering for the element establishes a "height" vector using the right hand rule that is in the opposite direction of the actual element height. In other words, the nodes should normally be ordered in a counter clockwise direction on the bottom surface of the element. Some codes such as Nastran, are insensitive to this ordering. If element checks are run, then Salinas will correct (and report) any solid elements found to have negative volumes. Without these corrections, the code will continue, but results that depend on these elements are suspect.

It is strongly recommended that any exodus file with negative volumes be corrected.

2.8.6 Displacement

Option **disp** in the **OUTPUTS** section will output the displacements calculated at the nodes to the output exodus file. The output file has the following nodal variables.

Variable	Description
DispX	X component of displacement
DispY	Y component of displacement
DispZ	Z component of displacement
RotX	Rotation about X
RotY	Rotation about Y
RotZ	Rotation about Z

In addition, if the analysis involves complex variables (currently `ceigen` as described in section 2.1.3), then the imaginary vectors are also included. They append "i" to the names above, e.g. the imaginary component in the X direction is "`DispXi`".

2.8.7 Velocity

Option **velocity** in the **OUTPUTS** section will output the velocities at the nodes to the output exodus file.

2.8.8 Acceleration

Option **acceleration** in the **OUTPUTS** section will output the accelerations at the nodes to the output exodus file.

2.8.9 Strain

Option **strain** in the **OUTPUTS** section will output the strains for all the elements to the output exodus file.

The following strains will be output for shell elements:

SStrainX1, SStrainY1, SStrainXY1 - *strains in the top layer of the shell*
SStrainX2, SStrainY2, SStrainXY2 - *strains in the mid-plane of the shell*
SStrainX3, SStrainY3, SStrainXY3 - *strains in the bottom layer of the shell*

Note: the top layer of the shell is determined by the ordering of the nodes of the shell. Also, the strains are in the local element coordinate system defined by the ordering of the nodes.

The following strains will be output for volume elements:

`VStrainX, VStrainY, VStrainZ, VStrainYZ, VStrainXZ, VStrainXY`

Note: These strains are in the global coordinate system, not the local coordinate system.

For more information on stress/strain recovery, see section 4.

2.8.10 Stress

Option **stress** in the **OUTPUTS** section will output the stresses for all supported elements to the output exodus file. Only shell and volume elements are supported, i.e. there is no stress output for beams.

Shell Stresses

The following stresses will be output for shell elements.

SStressX1, SStressY1, SStressXY1, SvonMises1 - *top layer of the shell*
SStressX2, SStressY2, SStressXY2, SvonMises2 - *mid-plane of the shell*
SStressX3, SStressY3, SStressXY3, SvonMises3 - *bottom layer of the shell*

Note: the top layer of the shell is determined by the ordering of the nodes of the shell, and can be output by using the **EOrient** *output options (see section 2.8.23). Also, the stresses are in the local element coordinate system defined by the ordering of the nodes.*

Volume Stresses

For volume elements, the stress is always output in the global coordinate system, not the local coordinate system. The following stresses will be output for volume elements:

Variable	Value
VStressX	σ_{xx}
VStressY	σ_{yy}
VStressZ	σ_{zz}
VStressYZ	σ_{yz}
VStressXZ	σ_{xz}
VStressXY	σ_{xy}
VonMises	von mises stress

For more information on stress/strain recovery, see section 4.

2.8.11 VonMises

Option **VonMises** in the **OUTPUTS** section will output the von Mises stress for all the elements to the output exodus file. For volume elements, the output will be the von Mises stress of the element. Surface elements define stresses on the top, center and bottom layers. The output will be the maximum of these 3 values.

Note that the von Mises stress is computed and output as a portion of the output if full stress recovery is requested. This option provides a mechanism for reducing output. Thus, if full stress output is requested, then the **VonMises** will provide no additional output. In other words, specifying both **VonMises** and **stress** in the outputs section is redundant, but does not result in an error.

2.8.12 VRMS

Option **vrms** will output computed root mean squared (RMS) quantities from a random vibration analysis. These quantities are written to a separate output file. Quantities output include the RMS displacement, acceleration and von Mises stress. In addition for the SVD option, the D matrix terms which contribute to the von Mises stress are output** (see section 2.1.14).

2.8.13 Energy

Option **energy** in the **OUTPUTS** section will place strain energies and strain energy density in the output exodus file. Note that the current implementation of strain energies requires recomputation of the element stiffness matrix, which can be expensive.

2.8.14 GEnergies

Option **GEnergies** in the **ECHO** or **OUTPUTS** section will trigger computation of global energy sums for the results and output exodus file, respectively. For the **ECHO** case, the computation includes the following.

strain energy The strain energy is computed from $u^T K u/2$ where u is the displacement and K is the current estimate of the tangent stiffness matrix. Note that this may not be complete for nonlinear solutions. Linear viscoelastic materials have contributions that will not be included in this sum.

kinetic energy Computed as $v^T M v/2$. Here v is the velocity and M is the mass matrix.

work The **work** is defined as,

$$W(t) = \int_{x(0)}^{x(t)} F(x) dx$$

**For a definition of D, see Reese, Field and Segalman.[3]

where F is the force and dx is the distance traveled. This can be restated as an integral over time.

$$W(t) = \int_0^t F(\tau)v(\tau)d\tau$$

where $v = dx/dt$ is the velocity. We approximate this at discretized time t_n as,

$$W_n \approx \sum_i^n F_i v_i \Delta t$$

Note that this is a sum over time using the simplest method possible. Because of integration error, it may not be completely consistent with the other energies above. For the **OUTPUTS** case, the total energy is written out at each time step.

2.8.15 Mesh_Error

The **mesh_error** keyword causes mesh discretization error metrics to be computed. These are computed as output quantities, but the overhead associated with the metrics is not negligible. Mesh discretization quantities depend upon the solution type, and are not available for all solutions. Output is typically available as element quantities (usually in the *mesherr* field). For some mesh discretization errors, a global quantity is also output.

Output	Description
ErrExplicitLambda	Relative error in λ.
ErrExplicitFreq	Frequency error estimate (Hz)

We note that for eigenvalue analysis, *relative* errors are reported for the eigenvalue when using the **mesh_error** keyword. Thus, for a given eigenvalue λ, the reported error is

$$\text{ErrExplicitLambda} = \frac{\lambda_h - \lambda}{\lambda} \tag{19}$$

This is more convenient since the analyst does not have to divide by the eigenvalues to see the percent error. The global variable "ErrExplicitFreq" provides an absolute estimate (useful in plots for example).

2.8.16 Harwellboeing

Option **harwellboeing** in the **OUTPUTS** section will output the mass and stiffness matrices in Harwell-Boeing format to files with *.hb* extension.

2.8.17 Mfile

Option **mfile** will cause **Salinas** to output various Mfiles like *Ksrr.m*, *Mssr.m*, etc. These files are mainly used by the **Salinas** developers for code maintenance and verification. Since many of these files can be quite large, caution should be exercised when using this option on large models. An index of some of the files written using this option is provided in Table 12.

2.8.18 Force

Option **force** in the **OUTPUTS** section will output the applied force vector to the output exodus file.

2.8.19 rhs

Option **rhs** in the **OUTPUTS** section will output the Right Hand Side vector from the calculations. For statics and dynamics, we repeatedly solve equations of the form, $Ax = rhs$. The **rhs** vector contains the applied forces and pressures as well as the inertial forces. Pseudo forces introduced in preload (say by TSR) are also part of this vector. This output is useful primarily for verification and debugging purposes.

2.8.20 EForce

Option **eforce** in the **OUTPUTS** section will output the element forces for line elements (such as beams and springs) to the output exodus file. Each two node, 1-dimensional element will have 3 force entries for each node, for a total of 6 element forces per element.

The element force is not a stress or a strain, and should not be used as such. If you want beam stresses, you must mesh that portion of the structure either as a shell or a solid. We output no stresses or strains for beams. EForce is used primarily to help understand the behavior of nonlinear line elements such as the Joint2G element (see section 3.25). The output is actually the direct output of our internal force routine (which is a nonlinear routine). It can be quite confusing to output these nonlinear forces in a linear analysis.

NOTE: The force returned is in the element (not global) coordinate frame. No provision is made for output of moments.

Table 12: Data Files Written Using the Mfile Option

Filename	**Description**
Stiff.m	Unreduced stiffness matrix including all active dofs
Kssr.m	Reduced stiffness matrix
Mass.m	Unreduced mass matrix
Mssr.m	Reduced mass matrix
LumpedMass.m	unreduced lumped mass matrix
xxx_gid.m	global IDs of the nodes
Fetimap_a.m	Map to convert from G-set to A-set The right hand side is the equation number. The lhs index is 7*(node index)+coordinate
Dampr.m	unreduced damping matrix (real components)
Dampi.m	unreduced damping matrix (imaginary components)
xxx_accelNN.m	G-set acceleration output of step NN
xxx_accel_aNN.m	A-set acceleration output of step NN
xxx_afNN.m	G-set applied force output of step NN
xxx_af_aNN.m	A-set applied force output of step NN
xxx_dispNN.m	G-set displacement output of step NN
xxx_disp_aNN.m	A-set displacement output of step NN
xxx_presNN.m	G-set nodal applied pressure of step NN
xxx_pres_aNN.m	A-set nodal applied pressure of step NN
xxx_velocNN.m	G-set velocity output of step NN
xxx_veloc_aNN.m	A-set velocity output of step NN
modal_amp.m	modaltransient output of mode amplitude vs time
ModalFv.m	modaltransient output of generalized forces

- The `xxx` above refers to the input file name root.
- The G-set output is 7*(number of nodes).
- The file names above are for the serial version of Salinas. In the parallel version, an underscore and the processor number will precede the ".m". For example, the reduced stiffness matrix becomes `Kssr_0.m`. There is no output of a globally assembled, parallel matrix - it does not exist.
- Some solution methods will not write all files. For example, there are no mass matrices output in the solution of statics. Generally, matrices are output in sparse symmetric row format.

2.8.21 Residuals

For most solution types, a linear solver is used to compute systems of the form $Ax = b$. For direct serial solvers, these systems are typically solved to numerical precision. However, with iterative solvers the solution is only approximate. Sometimes it is advantageous to evaluate the performance of the solver. For example, regions with large residuals may be candidate areas for mesh refinement, or may point to other mesh problems.

Eigen. For eigen analysis, the residual is $(K-\lambda_i M)\phi$. The vector is *not* normalized by the norm of ϕ, or any other quantity. A nodal residual work is also output. This is the product $\phi^T(K - \lambda_i M)\phi$ summed to the nodes, i.e. on a given node we sum the contributing degrees of freedom. Again, the value is *not* normalized. Clearly with mass normalized eigenvectors (which do not have units of length), the units of the residual work are not energy, and the term may well be negative. The residual is output for each mode.

Transient Dynamics. For transient analysis the residual reported is $Au-b$, where A is the dynamics stiffness matrix (see section 1.1 of the theory manual). With a displacement based Newmark-Beta integrator the dynamic stiffness is $K + \frac{2}{\Delta t}C + \frac{4}{\Delta t}M$. The residual is output at each time step.

In addition to the residual vector, the norm of the residual is output as a global variable.

2.8.22 Resid_only

It is occasionally useful to examine the residual after each solve. In the case of non-linear transient, or of eigen analysis, there may be many solves per output. Because of limitations in the output database format, it is very difficult (or impossible) to intersperse the residuals from each solve with the usual solution output. However, if all other output is turned off, we will write each residual to the output exodus file.

NOTE: This is really a debugging function. As such, we do minimal checks. It is possible to output displacement and resid_only. However, the eigenvectors will normally be properly written to the first m steps (where m is the number of modes), and the residuals will be written once per solve. There is no clear way to relate the residuals with the eigenvectors.
NOTE: For eigen analysis, it is possible to output the **resid_only**, *and the applied* **forces**. *This is the only time that forces make sense in an eigen analysis. These are really the load vectors provided by the iterative eigenvalue scheme (***ARPACK***).*

2.8.23 EOrient

Option **eorient** in the **OUTPUTS** section will output the element orientation vectors for all elements. The element orientation is a design quantity that normally does not change significantly through the course of an analysis. This output is provided to help in model construction and debugging.

The orientation vectors are output as nine variables that collectively make up the three vectors required for element orientation. The output variables and the associated meanings for various elements are shown in tables 13 and 14 respectively.

Table 13: Element Orientation Outputs

Name	Description
EOrient1-X	first orientation vector
EOrient1-Y	
EOrient1-Z	
EOrient2-X	second orientation vector
EOrient2-Y	
EOrient2-Z	
EOrient3-X	third orientation vector
EOrient3-Y	
EOrient3-Z	

2.8.24 Pressure

Option **pressure** in the **OUTPUTS** section will output the applied pressure to the output exodus file as an element variable. Note that there is no output for different sides of an element. Thus, if there is pressure applied to more than one

Table 14: Element Orientation Interpretation

Element	**EOrient1**	**EOrient2**	**EOrient3**
Beam2	axial	first bending (I1)	2nd bending (I2)
Shells	Element X	Element Y	Normal
Solids	Element X	Element Y	Element Z
HexShell	Element X	Element Y	thickness
ConM	NULL	NULL	NULL

face of an element, the output will represent only one of these pressures. Also note that the output provides a single pressure variable per element, and is not directly related to a particular element face. For most applications this provides a useful tool for checking input loads.

2.8.25 APressure

Option **apressure** in the **OUTPUTS** section will output the acoustic pressure to the output exodus file as a nodal variable. For purely acoustic elements, this will result in one degree of freedom per node, but for acoustic elements on the wet interface, this will result in four degrees of freedom per node in the output exodus file.

2.8.26 APartVel

Option **apartvel** in the **OUTPUTS** section will output the acoustic particle velocity to the output exodus file as an element variable. This is simply the velocity of the fluid particles. It is computed in Salinas as the gradient of the velocity potential. For purely acoustic elements, this will result in three degrees of freedom per element.

2.8.27 KDiag

Option **kdiag** in the **OUTPUTS** section will output the maximum and minimum values of the diagonal of the stiffness matrix as nodal variables KDiagMax and KDiagMin. These are the max and min of the 7 variables associated with the 3 translational, 3 rotational and 1 acoustic degree of freedom on each node. These values are primarily useful for diagnostics purposes, where they may help identify regions of a model that have extremely high stiffnesses. All 7 terms may be seen by outputting kdiag in the **ECHO** section.

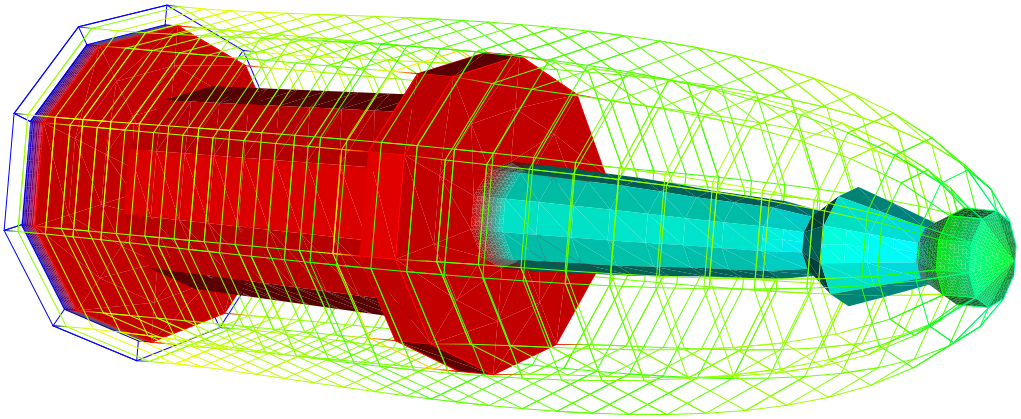


Figure 1: Example *KDIAG* output.

Figure 1 illustrates the use of this option. Note how the center sections of the model are highlighted by their stiffness terms. This tool is especially important for analyzing some collections of beams. Since beam stiffnesses are proportional to $1/L^3$, it is common to accidentally generate beams of extremely high stiffness, which can ruin the numerical solution.

2.8.28 Warninglevel

We have *partially* implemented some control over the output of warning messages. This is not implemented in general, but may be useful for some cases. The keyword **warninglevel** may be followed by either an integer, or a string.

level	descriptor	comment
0	none	minimal warning output
1	severe	only severe warnings output
2	bad	severe and bad warnings output
4	information	all warnings (default)

Table 15: OUTPUT Section Options

Option	Description
maa	mass matrix in the a-set
kaa	stiffness matrix in the a-set
faa	force vector in the a-set
elemqualchecks on ‖ off	default is on
ElemEigChecks	outputs first flexible eigenvalue, largest eigenvalue, and the ratio of the two for each element
disp	displacements at nodes
velocity	velocity at nodes
acceleration	acceleration at nodes
strain	strain of element
stress	stress of element
vonmises	vonmises stress on element
vrms	RMS quantities (random vibration only)
energy	element strain energy and strain energy density
genergies	global sum of energies
mesh_error	mesh discretization errors
harwellboeing	mass and stiffness matrices in Harwell-Boeing format
mfile	Outputs various Mfiles (mainly for developers)
locations	Outputs nodal coordinates and DOF to node map
force	Outputs the applied force
rhs	Outputs RHS of system of equations to be solved
pressure	Outputs pressure load vector
eforce	Outputs element forces for beams
eorient	Outputs element orientation vectors
warninglevel	Contol of warning messages

2.9 HISTORY

All the results from the "OUTPUT" section can be output to a limited portion of the model using a history file. Only those outputs described in Table 15 are supported. Note that if the output is also specified in the OUTPUT section, there is little need to write the data in the history file. The following output section options are ignored in the history section because all history file output will be in the exodus format.

- mfile
- harwellboeing
- kaa, maa, faa
- vrms

In addition to the **output** selection options, the history file section contains information about the regions of output. The default is NO output selection. Selection may be for node sets, side sets, a node list file (see section 2.12.3), or element blocks. If side sets are selected,the side set selection is for the nodes associated with that side set, not for the elements themselves. All nodal variables selected in the history file will be output for all selected nodes. Selecting an element block automatically selects the associated nodes in that block. The format for the selection is the same as that of the subdomains selection in the ECHO (section 2.7). For example,

```
HISTORY
   nodeset '1:10,17'
   sideset  '3:88'
   nodeset '8,15' coordinate 4
   block 5,6,3
   stress
   disp
END
```

Any number of **nodeset** selections can be specified in the history section. Nodeset specifications may be followed by an optional coordinate entry. If a **coordinate** is specified, all nodal results for the nodes in the nodeset are transformed to the specified coordinate system before output to the file. If a particular node is identified in more than one specification, the last specification is used for the output. The coordinate ID of nodes in the history file may be printed out in the echo file by specifying **nodes** in the **echo** section of the input. The coordinate ID will also be

written to the history file (as a nodal variable *CID*) provided any nonzero coordinate frames have been specified.

Only one **block** and one **sideset** specification is permitted in the history section. Output coordinate frames may only be specified on nodesets.

Unlike subdomains, node set and side set IDs need not be contiguous in the exodus file. The selection criteria may identify nonexistent sets. These will be silently ignored. In the above example, if the input exodus file contains no node set with ID=10, it will not be treated as an error. Node set and side set IDs in the history file will be consistent with the corresponding exodus input file.

Only one history file will be written per analysis. The name of the history file is derived from the name of the exodus output file, except that the extension is ".h".

While the history file provides a convenient means for transforming coordinates, its applicability may be somewhat limited when output in many coordinate frames is desired. In particular, only a single history file is written in each analysis, and only one coordinate frame may be output per node. See the **coordinate** section (2.22) for information on obtaining the transformation matrices from each coordinate frame directly.

2.10 FREQUENCY

The frequency section provides information for data output from the modalFRF, directFRF, shock, modalshock, and random vibrations solution methods. One frequency file is written per analysis. The name of the frequency file is derived from the name of the exodus output file, except that the extension is ".frq". The section format follows that of the history section, except that currently the only outputs are nodal variables. Elements will be collected and placed in the geometry definition of the file, but only nodal variables are output at each frequency value. Solution methods that do not write frequency domain output silently ignore the Frequency section.

The frequency section also includes the definitions of the frequency range and step. (In the beta release notes, these were included in the solution section).

A frequency section (with some output selection region) must be selected for any solution method requiring frequency output. To fail to do so is an error, since the solution would be computed and no output provided.

```
FREQUENCY
   nodeset '1:10,17'
   sideset  '3:88'
   block 5,6,3
```

```
    disp
    acceleration
    freq_min=10    // starting frequency in HZ
    freq_step=10   // frequency increment
    freq_max=2000  // stop freq. This example has 201 frequency points.
END
```

The controls in the **frequency** section also affect data written to the results (or echo) file. In particular, the echo file contains data only for those nodes in the selection region of the **frequency** section. Selection of a specific output (such as displacement or acceleration) is independent. For example, you may echo only displacements, but write displacements and accelerations to the exodus frequency output file.

The *seacas* translator *exo2mat* may be used to translate the output into matlab format for further manipulation and plotting.

2.11 FILE

Disk files names are specified in the **FILE** section. The parameters for the **FILE** section are,

Option	Description
geometry_file	Indicates which **Exodus** file to use
numraid	Indicates how many raids are available (for parallel execution)
linesample_file	Indicates which linesample (text) file to use
sierra_input_file	optional file name for transfer of data from a sierra application

2.11.1 geometry_file

The geometry file is used for input of the mesh geometry including the nodes, elements, connectivity and attributes. It is typically a binary exodus file.

In an MP environment, the file name is determined by the number of raid controllers and the processor number. The actual file name is computed by this command:

```
sprintf(filename,fmt, (my_proc_id%numraid)+1, my_proc_id );
```

where fmt is the string specified for the geometry file. The number of raid devices is defined using the keyword **numraid**. For example, on a single processor, a **FILE** section may look like this.

```
FILE
    geometry_file 'exampleg.exo'
END
```

On multiple processors this might look like:

```
FILE
  geometry_file '/pfs_grande/tmp_%.1d/junk/datafile.par.16.%.2d'
  numraid 2
END
```

This will result in opening these files:

```
/pfs_grande/tmp_1/junk/datafile.par.16.00
/pfs_grande/tmp_2/junk/datafile.par.16.01
/pfs_grande/tmp_1/junk/datafile.par.16.02
/pfs_grande/tmp_2/junk/datafile.par.16.03
/pfs_grande/tmp_1/junk/datafile.par.16.04
...
/pfs_grande/tmp_2/junk/datafile.par.16.15
```

Note that if the file name is not included in quotes, it will be converted to lower case. Appendix C shows the steps involved in the parallel execution of **Salinas**.

2.11.2 Linesample

Line sampling provides a means of evaluating fields or internal variables at sampling points within a structure. These sampling points are defined on a series of lines.

The primary application is verification of stress fields read into Salinas from TSR (see section 2.1.25).

The format of the linesample file is as follows:

- line 1: an integer listing the number of samples per line. Each line will have this number of equally spaced samples. each subsequent line contains 6 real numbers. These represent the coordinates of the beginning and endpoints of the sampling line.

- Each subsequent line has the 6 real numbers. These represent the coordinates of the beginning and endpoints of the sampling line. Any number of these lines may be provided.
- Terminate with EOF.

The output will be written to a **matlab** m-file with the name provided by the argument to **linesample**. One file is written per analysis (results are joined analogous to history file output).

2.11.3 sierra_input_file

The **sierra_input_file** may be used as a restart following a sierra calculation (using **Presto** for example). This is an alternative to directly transferring the same data using the sierra transfer services. The **sierra_input_file** has the same format and usage as **geometry_file**, and can be used to transfer data in parallel or serial. See also section 2.1.19.

2.11.4 Additional Comments About Output

A text log or *results* file can be written for the run. Details of the contents of the results file are controlled in the **ECHO** section (see section 2.7). The results file name is determined by the name of the input file, and will be in the same directory as the input text file, regardless of whether **Salinas** is being executed in serial or parallel. However, if executing in parallel, using the "subdomains" option in the **ECHO** section allows control of the number of *results* files. For example, if running on 100 processors, up to 100 result files may be output. Using **subdomains** "`0:2`" will only output three files, from subdomains 0, 1, and 2. The default is to output a *results* file only for processor zero. The results file name uses the base name of the input, with an extension of ".rslt". In a parallel computation, the results file names use the base name of the input file, followed by an underscore and the processor number, then followed by the "`.rslt`" extension.

For calculations in which geometry based output requests are included (see section 2.8), an output **Exodus** file will be created. The **Exodus** file is a binary file containing the original geometry plus any any requested output variables. The output **Exodus** file name is determined from the geometry file name. The base name of the output is taken from the geometry file by inserting the text "`-out`" just before the file name extension. The output **Exodus** file will be written to the same directory where the geometry file is stored. If executing **Salinas** on a parallel machine, the **Exodus** output files should be written to the raid disks for reasonable performance.

2.12 BOUNDARY

Boundary conditions are specified within the **Boundary** section. Node sets, side sets or nodelists may be used to specify boundary conditions. By default, they are specified in the basic coordinate system, but an alternate system may be specified using the "coordinate" keyword (see section 2.22). The current implementation is very inefficient if any but the basic system is used (they are treated as MPCs). The following example illustrates the method.

```
BOUNDARY
  nodeset 1
       x = 0.1     // constrain x=0.1 for all nodes in set
       y = 0       // constrain y=0 throughout nodeset 1
       RotZ = 0    // constrain the rotational dof about Z
  nodeset 2
    fixed          // constrain all structural dofs in nodeset 2
  nodeset 3
    accelx = 0.3   // constrain the x component of acceleration,
    function=1     // in nodeset 3, with the time-dependence
    disp0 = 0.0    // given by function 1, and initial conditions
    vel0 = 0.1     // given by disp0, vel0
  sideset 3        // acoustic sideset
    pdot = 1.0     // constrain the time derivative of acoustic pressure
    function = 2   // in sideset 3, with the time-dependence
    p0=1.0         // given by function 2, and initial condition p0
  sideset=4
    coordinate 7   // apply in coordinate system 7
    x=0            // constrain only the X direction in frame 7
  node_list_file='clamped.nodes'
    fixed
END
```

The descriptors for the displacement boundary conditions are, **X**, **Y**, **Z**, **RotX**, **RotY**, **RotZ**, **P**, and **fixed**. An optional equals sign separates each descriptor from the prescribed value. The value **fixed** implies a prescribed value of zero for all degrees of freedom.

2.12.1 Prescribed Displacements

In linear statics, one may prescribe a nonzero displacement by entering a value following the coordinate direction. In the example above, the displacement for nodeset 1 is set to 0.1 in the X direction.

For linear statics, there must be no **function** entry following the entry. Prescibed displacements have the same limitations as prescribed accelerations described in the next section.

The load in this case is introduced by the prescribed displacement. However, the **loads** section must exist (for error checking purposes) even if it is empty.

2.12.2 Prescribed Accelerations

In transient dynamics, the acceleration on a portion of the model may be prescribed as a function of time. The descriptors for prescribed accelerations are, **accelX**, **accelY**, **accelZ**, **RotaccelX**, **RotaccelY**, **RotaccelZ**, **Pdot**. A function must be used to apply the time-dependent boundary accelerations. Optional initial displacement and velocity can also be specified; if not, they default to 0. In the example above, the x acceleration of nodeset 3 will be prescribed as $0.3 \times f(t)$, where $f(t)$ is defined in function 1. The initial displacement is given as 0, and the initial velocity is 0.1. Currently, only accelerations can be prescribed. However, this does not preclude problems with prescribed velocities and displacements, since these cases can be converted to a prescribed acceleration by differentiation. Note that if no function is listed, an error message will be generated.

We note that in the case of an acoustic sideset or nodeset, the prescribed value is the first time derivative of acoustic pressure, denoted above as **Pdot**. This is because, internally, Salinas solves for the velocity potential, and the first time derivative of the velocity potential is the acoustic pressure. Thus, by specifying the first time derivative of pressure, one is actually prescribing the acceleration of the velocity potential.

An additional point to consider when applying prescribed accelerations is that the initial velocity and displacement (denoted as **disp0** and **vel0**), are also necessary to completely define the boundary condition. These values account for the constants of integration obtained when integrating the prescribed acceleration to obtain the corresponding velocity and displacement on the sideset or nodeset. In the case of acoustics, only one initial condition is needed (**p0** which specifies the initial acoustic pressure), since only the first time derivative of acoustic pressure is specified. We note that **disp0**, **vel0**, and **p0** all default to 0 if not specified.

There are some limitations with the prescribed acceleration capability, which we list now. First, prescribed accelerations are not currently set up to work with multicase solutions. Also, they only work in the standard (Cartesian) coordinate system. Prescribed accelerations can be used in meshes that have nonlinear or viscoelastic elements, as long as the prescribed accelerations are not applied directly to the nonlinear or viscoelastic elements. Lastly, we note that the nodes involved in prescribed accelerations cannot coincide with nodes that are involved with mpcs.

Finally, we note that when prescribed accelerations are used, they induce a load on the structure. Thus, in many cases the **loads** section serves no purpose, unless an additional external load is applied. In these cases, however, we note that that an empty **loads** block is still needed in the input file. An error message will be generated if the input file has no **loads** section.

2.12.3 Node_List_File

To make it a little easier to apply boundary conditions, a *node_list_file* option is provided. In this option, an additional text file is provided which contains a list of global node ids separated by white space. No comments, or other characters are allowed in the file, as shown above. The remainder of the boundary condition specifications are unchanged.

There are several limitations place on collections of nodes specified in this manner.

1. This is a rather inefficient method of supplying the nodes. It is recommended that nodesets or sidesets be employed when practical.
2. No node distribution factors may be provided.
3. Only one *node_list_file* can be provided per section, e.g. there may be only one such file in a *boundary* section.
4. The output exodus file will have no record of this list.
5. The global node numbers are the unmapped exodus indices. This means that the numbers go from 1 to N, where N is the maximum number of nodes in the model. This definition is the only one which allows the same node numbering to be used in both a serial and parallel file.
6. There is NO requirement that the nodes be sorted in the list, but repeating a node in the list can have undefined results, i.e. don't do it.

2.13 LOADS

Loading conditions are specified within the **loads** section. The following example illustrates the method.

```
LOADS
   nodeset 3
      force = 1.0 0. 0.
      scale = 1000.
```

```
      function = 2
   nodeset 5
      force = 0. -1 0
   body
      gravity
      0.0 1.0 0
      scale -32.2
   body
      thermal_load
      function = 1
   sideset 7
      pressure 15.0
   sideset 12
      traction = 100.0 20.0 0.0
   sideset 13
      acoustic_load 1.0
      function = 1
   node_list_file='force.nodes'
      force=1.0 0 0.
      scale = 100.
      function=2
END
```

Loads may be applied to node sets, side sets, node lists (see section 2.12.3) or the entire body (in the case of inertial loads). Loads are applied in the global coordinate system using nodesets. Pressure loads may be applied using side sets. The pressure is always normal to the surface. All loads applications are additive.

The syntax followed is to first define the region over which the load is to be applied (either **nodeset**, **sideset**, **node_list_file** or **body**). Each such region defines a *load set*. For each such definition, one (and only one) load type may be specified. However, any region definition (except **node_list_file**) may be repeated so that forces and moments may be applied using the same node set. The load types are,

Option	Parameters
force	*val1 val2 val3*
moment	*val1 val2 val3*
gravity	*val1 val2 val3*
pressure	*val1*
acoustic_load	*val1*
traction	*val1 val2 val3*
thermal_load	*none*

Following the definition of the load type, a vector (or scalar in the case of pressure loads) must be specified, except in the case of a thermal load, where no vector or scalar multiplier is needed. The total force is the product of the load vector, the scale factor, and the nodeset distribution factor found in the exodus file. Note that in some cases the nodeset distribution factor may be zero. In that case, the total applied force will also be zero. The **pressure** and **acoustic_load** loadings may only be applied to side sets. The total pressure is the product of the scale factor, pressure (scalar) and sideset distribution factors. If the pressure loading in NOT normal to the sideset, the **traction** capability should be used. NOTE: Pressure will act on a surface in a compressive sense, while a traction can be specified as any vector which will act on the **sideset** specified in the direction given by the triple values specified after **traction**. Also, traction loads are applied on the faces of the shell elements in a piecewise manner, i.e., the traction load acting on a face of the element is assumed constant. If the distribution factors on the nodes of the element vary, the average of the load (element per element) is assumed.

The **acoustic_load** loading may only be applied to acoustic elements. It specifies the fluid velocity in the normal direction only, since for inviscid fluids the tangential velocity components cannot be 'forced'. Note that this is the counterpart to the **pressure** load for structures, in the sense that it is a Neumann boundary condition.

Variation of the load over space is accomplished using node set or side set distribution factors. If these are provided in the **Exodus** file, the load set is spatially multiplied by these factors. The total loading is the sum of the loads for each load set summed over all the load set regions.

Finally, we note that when prescribed accelerations are used, they induce a load on the structure. Thus, in many cases the **loads** section serves no purpose, unless an additional external load is applied. In these cases, however, we note that that an empty **loads** block is still needed in the input file. An error message will be generated if the input file has no **loads** section.

2.13.1 Thermal Loads

The **thermal_load** option is used in conjunction with a spatial temperature specification for the structure. The temperature distribution can either be specified via the input exodus file, or on a block-by-block basis, as described below. Based on the temperature distribution, a thermal load is computed and then applied to the structure.

If the solution method is selected to be statics, the **thermal_load** option will provide the thermal load necessary to solve the thermal expansion problem. If the solution method is transient dynamics, the same thermal load will be applied as in the statics case, but modulated by the function that is specified below the **thermal_load** keyword. This corresponds to a thermal shock analysis. Thus, for a transient dynamics problem that includes damping, and with a function that is equal to 1.0 for all time, the transient analysis would eventually converge to the same solution as obtained in the statics analysis, which would be the solution from a classical thermal expansion analysis.

The temperature field can either be read from an exodus file, which would typically be the result of a Calore analysis, or it can be specified on a block-by-block basis in the input deck. For temperature fields that change from element to element, the temperatures must be read in from the Calore output file. For more uniform temperature distributions, it is more efficient to specify them block-by-block in the input deck.

Note that when using thermal loads, the temperature data is expected to either be in the mesh (exodus) files, or specified using the input deck (i.e. block-by-block). The format of the exodus file output is the same as that generated by Calore, as described below.

If temperatures are specified using the input deck, then each block must be given its own temperature. In the example below, there are 2 blocks, and each is given a different temperature.

```
BLOCK 1
  material 1
  T_current 100
END
BLOCK 2
  material 2
  T_current 200
END
```

Note that the default for $T_{current}$ is 0.

When temperatures are read in from a Calore output file, the material properties can be specified as temperature-dependent. This implies that each element will have different material properties. More details are given in the section on temperature-dependent material properties.

For thermal statics or thermal transient analysis, each material block must be given two additional parameters, the reference temperature, $Tref$, and the coefficient of thermal expansion, $alphat$. These parameters are defined via the thermal strain, which is given by

$$\epsilon_{thermal} = alphat\,(T_{current} - Tref) \tag{20}$$

An example is the following.

```
MATERIAL 1
        E 10e6
        nu 0.3
        tref 300.0
        alphat .001
        density 0.1
END
```

The defaults for $tref$ and $alphat$ are both 0.0. This implies that if they are not specified, then the material will not contribute to the thermal analysis (see equation 20).

Shell and beam type elements are not supported in thermal analysis. If used in conjunction with a thermal load, their contributions to the thermal expansion analysis will be ignored. This shortcoming is expected to be corrected in a future release.

The labels for the temperatures must be as shown in the table below. This is the output format used by Calore.

Name	Definition
TEMP	the nodal temperature

The **thermal_load** load case can be used in a multicase solution method. In that case, the stresses and internal forces from the thermal analysis are used as initial conditions for the next case. For example, for a fixed-fixed cantilever beam that is subjected to a uniform temperature increase, the beam will undergo a stretch due to the thermal static analysis, and will have residual stresses. If this beam were then subjected to an eigen analysis in a subsequent case, the modes would be modified due to the geometric stress stiffening. Conversely, for a fixed-free beam, there would be no residual stresses and thus no effect on subsequent cases. Note that

the displacements from thermal analysis are not carried over to subsequent cases. Thus, to get the total displacement from a thermal analysis followed by transient, one would need to add the displacement results from the two cases separately.

When reading in temperature data for use in a thermal analysis, there is one additional input needed in the **PARAMETERS** block, the keyword **thermal_time_step**. The following gives an example.

```
PARAMETERS
  thermal_time_step 10
END
```

The Calore output files can contain multiple time steps of data. The user can select which time step is to be used for defining temperature data in Salinas, using the keyword **thermal_time_step**. In this case the tenth time step will be read in from the Calore output files. The default value for keyword **thermal_time_step** is 1.

The following is an example of some of the input for a thermal statics analysis.

```
SOLUTION
  statics
END

PARAMETERS
  thermal_time_step 10
END

LOADS
  body
    thermal_load
END
```

2.13.2 Consistent Loads

The loads for all of the 3-D and 2-D elements are calculated in a consistent fashion when a pressure load is applied. For more details on the implementation, see the

programmer's notes. It is very important that consistent loading be used. This is especially true for shell elements where the consistent loading is required to properly apply rotations.

2.13.3 Time Varying Loads

Additional options provide the capability of varying the load over time. The **loads** options include,

- **scale** with one parameter provides a scale factor to be applied to the entire load set. Only one scale may be provided per load set.
- **function**. A time varying function may be applied by specifying a function ID. Only one function may be applied per load set. The function is defined in the **function** section (see section 2.23 on page 106). The loads applied at time t for a particular load set will be the sum of the force or moment vectors summed over the nodes of the region and multiplied by the scale value and the value of the time function at time t.

NOTE: If a function is defined for a particular load, then it is assumed to be a transient load. If there is no function defined then it is assumed to be a static load. The solution procedure chosen will only use the loads that are applicable, i.e., a static solution will only use static loads and a transient solution will only use transient loads.

2.13.4 Frequency Dependent Loads

Frequency dependent loads may be applied in frequency response analysis. The real part of these loads is applied exactly as above with the understanding that the functions referenced now apply to frequency not time. Frequency dependent loads may include an imaginary component. This is done by prefixing the load types listed above by the letter *"i"*. Thus the imaginary part of the load uses these load types.

For Complex Analysis

Option	Parameters
iforce	*val1 val2 val3*
imoment	*val1 val2 val3*
igravity	*val1 val2 val3*
ipressure	*val1*
itraction	*val1 val2 val3*

A function must be associated with each such load. An example follows.

```
LOADS // example for FRF analysis
  nodeset 1
     force=1 0 0      // the real part of the load
     function=11
  nodeset 2
     iforce=1 0 0     // the imaginary part of the load
     scale .707
     function=12
END
```

2.14 Load

Loading conditions *for all multicase solutions* are specified within the **load** section. See paragraph 2.1.1 for information on specifications for multicase solutions. The **load** section is *identical* to the **loads** section described in the previous paragraph (2.13), except the the section begins with the **load**, and a load step identifier is required. The following example illustrates the required input.

```
LOAD=57
   nodeset 3
      force = 1.0 0. 0.
      scale = 1000.
      function = 2
   nodeset 5
      force = 0. -1 0
END
```

Unlike the **loads** section, there may be multiple **load** sections in the file, with each entry corresponding to an applicable step in the solution.

2.15 RanLoads

The **RanLoads** section is used to provide input information for spectral input to a random vibration analysis. Note that this input will contain both a spatial and temporal component. The **RanLoads** section contains the following required keywords.

Parameter	Argument	Description
matrix	*Integer*	matrix-function identifier
load	*Integer*	row/column identifier

The **matrix** keyword identifies the appropriate **matrix-function** (see section 2.24). The matrix-function determines the dimensionality of the input (using the **dimension** keyword). It also determines the spectral characteristics of the load.

The spatial characteristics are determined in **load** sections within the **RanLoads** definition. There must be exactly as many **load** sections as the dimensionality of input. For example, if the S_{FF} matrix is a 3x3, then there should be 3 separate load sections. Each load section within the **RanLoads** block must be followed by an integer indicating to which row/column it corresponds. The details of each **load** section are identical to the over all **loads** section (see 2.13) except that no time/frequency function is allowed. Note that only one load is required per row of the S_{FF} matrix, but each *entry* of the matrix may have a spectral definition (identified by a real and/or imaginary **function**).

The following example illustrates the definition of a single input specification. The loading is scaled so that a 1000 lb mass located on the input point (in nodeset 12 here) is scaled to produce a unit g^2/Hz loading.

```
RANLOADS
   matrix=1
   load=1
     nodeset 12
        force=0 1 0
        scale 1.00e3  // needed to convert to g
        // loads input in lbs. The PSD is in g^2/Hz.
        // F = accel * mass
        //   = accel * (scale_factor)
        //   = accel * ((1000*.00259)*384.6)
END
```

Scaling the input force for a random vibration analysis can be confusing. This is especially true since enforced acceleration cannot be used to apply the force. The example above applies to english units where a **wtmass** parameter has been applied. For SI units or other systems where wtmass=1, the force would need to be multiplied by g to apply the input as acceleration in g's.

The input acceleration may be examined by evaluating the output PSD at the input degree of freedom. This is done by putting the applied load set into the **frequency** section (2.10), and adding the **acceleration** keyword. The output is in the native units of analysis. For the example above, the output will be in $(in \cdot lbm/s^2)^2/Hz$, and must be divided by $(386.4)^2$ to convert to g^2/Hz.

2.16 Contact Data

Limited Node-to-Node contact is available in release 2.0 (February 2005). While some tied surfaces are in place (see section 2.17), general contact is unavailable in the current release. The following documentation is retained in the user's manual to facilitate feedback from our user community, but much of the functionality is still pending.

Contact surfaces provide the ability to model structures that may be in contact part of time. Examples include a tire rolling on the road, rattling in a joint and structures under impact. Note that a tied surface capability is available for the special case where the surfaces will always be in contact (see section 2.17). Contact surfaces are inherently nonlinear. The simplest of such structures are characterized by frictionless contact, i.e. a restorative force is provided only in the normal direction, and slippage occurs tangent to the normal. This is the only type of contact currently supported in Salinas, though frictional and sticking contact are under development.

Contact is specified as shown in the example below.

```
CONTACT DATA
    Surface 33,17
    Friction Static = 0.0
    Search Tolerance = 1e-8
    Interaction=node-to-node
END
```

The example above defines a region of contact between sidesets 33 and 17. While contact is defined here between sidesets, it is currently limited to node-to-node contact. The coefficient of static friction is defined to be zero (the default). The search tolerance sets the radius for search for the nodes on the surface. The relevant parameters for contact are shown in Table 16.

The only interaction method currently supported is "node-to-node" which is specified with the "interaction" keyword. Also, the only supported friction model is none, specified by a coefficient of static friction of zero. Contact is enforced within the solver, and is currently supported only when using the FETI-CF solver (Appendix E). Enforcement does not truly require distinguishing between the master and slave surface. The contact is enforced symmetrically. We have adjusted the input to be as consistent as possible with other Sandia codes.

2.17 Tied Surfaces

Tied surfaces provide a mechanism to connect surfaces in a mesh that will always be in contact. Because the surfaces are always tied, the constraints may be represented

Table 16: Contact Data Parameters

Parameter	type	description
Surface	*integer pair*	master and slave sideset separated by comma or space
Friction Static	*Real*	coefficient of static friction (defaults to zero)
Search Tolerance	*Real*	Radius of search for paired nodes defaults to 1e-8
Edge Tolerance	*Real*	search tolerance beyond an edge facet defaults to 1/10 search tolerance.
Interaction	*String*	node-to-node node-to-face *(not currently supported)* edge-to-face *(not supported)* face-to-face *(not supported)*

by a set of linear multipoint constraints (see Appendix 3.29). Tied surfaces are also known in the literature as "glued surfaces" or as "tied contact". They are used almost exclusively to combine two surfaces of a mesh that have not been meshed consistently.

There are a number of ways of combining surfaces that have not been consistently meshed. The simplest method constrains the nodes of the slave surface to lie on the master surface. In this method, the constraint is called *inconsistent* because the mesh does not ensure that linear stress will be maintained across the boundary. The stress and strain in the region of the constraint will be wrong. However, loads are properly transferred across the boundaries, so a few element diameters away from the boundary, the stresses and strains should be approximately correct.

We note that tied surfaces can currently be specified for structural-structural interfaces, acoustic-acoustic interfaces, and structural-acoustic interfaces (i.e. wet interfaces). The syntax in the **TIED DATA** block is the same in all three cases. In the first case, the nodal displacements on the slave surface are constrained to lie on the master surface. In the last case, the nodal acoustic pressures on the slave surface are constrained to match those on the adjacent master surfaces. In the case of tied structural-acoustic interfaces, it is necessary to insure a weak continuity of both stress and displacement (velocity) across the wet interface. A future version of the theory manual will describe this procedure in detail. Also for tied structural-acoustic interfaces, we recommend that the acoustic surface be defined as the master (and hence should have its sideset number listed first in the input deck). Defining the structural surface as the master sometimes causes an error related to singular

subdomain matrices.

We do allow mixing of tied surface cases in a given simulation. For example, one may have tied acoustic-acoustic and tied structural-acoustic data blocks in the same input file. However, it is necessary that each sideset be exclusively attached to either structural elements or acoustic elements. A single sideset cannot simultaneously contain both acoustic and structural elements. This does not restrict the types of analysis that can be done, but it may lead to more **TIED DATA** blocks. However, this extra input will reduce confusion and likely also reduce potential modeling errors.

Mortar methods may also be used to tied the surfaces. These methods are more tightly integrated with the solver, and are currently only available using the FETI-CF solver (Appendix E). The cost in computing the mortar contribution is higher than the *inconsistent* method, but the solution will typically be much better in the region of the constraint.

In the future, the *inconsistent* tied surface will be transitioned into a fully consistent algorithm. The default method will become "mortar".

Tied surfaces are specified by a listing of master and slave side sets. Any number of tied surfaces may be specified in the input, i.e. more than one tied surface section may occur in the input. Each tied surface section represents a *single* logical pairing of constraint side sets.

```
TIED DATA
    Surface 12, 18
    search tolerance = 1e-7
    edge tolerance = 1e-8
END
```

In the example above, sideset 12 is defined as a master surface. Side set 18 is the slave surface. Each node in the slave surface is tied to the set of nodes in the corresponding element face of the master surface.

Tied surfaces use a node-to-face search algorithm. In this algorithm, the "search tolerance" represents the normal distance from a node on one surface to a corresponding face on the other. Thus, the search tolerance will typically be quite small and represents the amount the two surfaces may not be coincident. This is in contrast to a node-to-node search, where the "search tolerance" represents a search radius. See Figure 2.

> *Note: The current implementation ties a master and slave surface that are* **face** *connected only. We have not implemented or tested a capability to tie the* **edges** *of shells.*

The relevant parameters for tied surfaces are shown in Table 17.

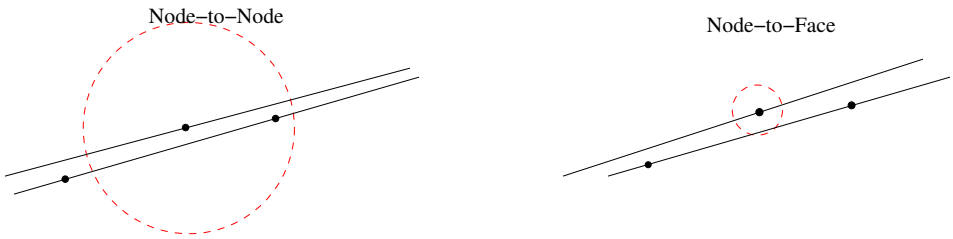


Figure 2: For node-to-node searches the *search tolerance*, must be large enough to capture nearby nodes. For node-to-face searches (as used in tied surfaces), it should only capture the nearby surface.

Table 17: Tied Surface Parameters

Parameter	type	description
Surface	*integer pair*	master and slave sideset separated by comma or space
Search Tolerance	*Real*	face normal of search tolerance defaults to 1e-8
Edge Tolerance	*Real*	search tolerance beyond an edge facet defaults to 1/10 search tolerance.
Interaction	*String*	node-to-node *(not supported)* node-to-face *(default)*
Method	*String*	inconsistent *(default most solvers)* mortar *(default for CF solver)*

2.18 RigidSet

Rigid Sets are intended as a usability tool to permit the analyst to treat a set of nodes as completely rigid. The input is straightforward.

```
RIGIDSET set1
     sideset 1
     sideset 2
     nodeset 88
END
```

The above definition would establish a single set that is tied together. For purposes of error reporting only, the name "set1" is associated with this example set. If multiple *independent* sets are required, then multiple rigidset definitions may be made.

The relevant parameters for **rigidsets** are shown in Table 18. Any number of `RigidSet` sections may be introduced, each will act independently. Exodus sideset or nodeset information may be included in the definition.

Table 18: RigidSet Parameters

Parameter	**type**	**description**
sideset	*integer*	sideset id
nodeset	*integer*	nodeset id

Limitations. Rigidsets meet an important need to tie many nodes together. Generally they are much more robust than generating collections of **RBAR**s or other rigid elements. However, it is very easy to generate redundant constraints through this input. Redundant constraints cause most linear solvers to fail, and we aren't good at providing diagnostics. Generally,

1. rigidsets must be completely disjoint, i.e. they may share no common nodes. If they share a node, they should be put in the same rigid set.
2. None of the nodes in the rigidset should be constrained (as through a boundary condition).
3. Other constraints (such as RBARS) should not further constrain the set.

This limitation does not prohibit the addition of an **RBAR** or other constraint which ties the rigidset to an *otherwise unconstrained* node.

2.19 BLOCK

Each element block in the **Exodus** file, must have a corresponding **BLOCK** entry in the input file. This section contains information about the properties of the elements within the block[††].

2.19.1 Block Parameters

There are two main types of block parameters:

1. Parameters exist which are common to most elements. These include:
   - Material property references are required for most elements. The material reference is of the form, `material=material_id`, where `material_id` is a string representing the material identifier (see section 2.21).
   - coordinate frames - *optional*
   - nonlinear behavior - *optional*
   - block damping - *optional*
   - non-structural mass - *optional*
2. Element specific names and parameters. These properties depend on the element type. Clearly shells will require a thickness, while it is meaningless for solids.

Block Example. An example is provided below.

```
// the following element block is Tria3
BLOCK 32
    material 2
    tria3
    thickness 0.01
END

// the following element block is hex.
// exodus tells us it is an 8-node hex.
// The only required argument is the material card
BLOCK 34
```

[††]It is not an error to have a block entry in the input file that has no corresponding exodus file entry. Such an entry will be silently ignored. It is an error to have multiple definitions for the same block. However, Salinas does not report the error. Which definition is used is not defined.

Table 19: General Block Parameters

Keyword	Values	Description
nonlinear	yes/no	blockwise nonlinear behavior
coordinate	*integer*	reference coordinate frame
blkalpha	*Real*	blockwise mass proportional damping
blkbeta	*Real*	blockwise stiffness proportional damping
nsm	*Real*	blockwise non-structural mass

```
        material aluminum
END

BLOCK 3
    Coordinate 1
    Spring
    Kx=1e6
    Ky=0
    Kz=0
    BLKBETA=0.0031
END
```

A list of the applicable attributes for some of the different element types is shown in Table 21. Each element type is outlined in section 3.

2.19.2 General Block Parameters

Parameters that are generally applicable to almost all blocks are listed in Table 19. More detailed descriptions are available in the following paragraphs.

Nonlinear Behavior. The nonlinear behavior of the block is controlled by the **nonlinear** keyword. The global default for block-level nonlinear behavior is set in the PARAMETERS section. Within each block, we can override that default value. For example, to set a block to default to linear behavior, we would have the following BLOCK definition.

```
BLOCK 3
```

```
      nonlinear=no
      material 2
      tria3
      thickness 0.01
END
```

Similarly, to turn on the nonlinear behavior for the block, we would have,

```
BLOCK 3
      nonlinear=yes
      material 2
      tria3
      thickness 0.01
END
```

Note that these block-level nonlinear flags override the global **nonlinear_default** keyword that is set in the PARAMETERS section.

Coordinate Frame Reference. The reference coordinate system may be defined in a block. This definition applies to all the elements of the block and the associated materials. At this point, the coordinate system is only recognized for a subset of the elements (solid elements and springs). Further information on coordinate systems may be found in section 2.22.

Block Specific Damping. In section 2.28, various methods of specifying the damping parameters for a model are identified. In addition to these methods, block specific damping parameters may be applied. These apply a stiffness (or mass) proportional damping matrix on an element by element basis within the block. Thus, if a model is made of steel and foam, one could apply a 5% stiffness proportional damping term to the foam, but leave the steel undamped.

There is no physical justification for proportional damping, and there is no expectation that it will accurately represent damping mechanisms in a structure. However, it is easy to apply, and there are cases where proportional damping may reveal a need for more accurate damping models. As with all damping models, the effects depend on the solution type. For example, both statics and eigen analysis ignore the damping matrix.

The damping matrix generated from block specific damping is defined as follows.

$$D = \sum_{i}^{nblks} \alpha_i M_i + \beta_i K_i \tag{21}$$

Table 20: Non-Structural Mass Units

Element Type	Units	Example
One Dimensional	mass/length	lbs / in
Two Dimensional	mass/area	lbs / sq-in
Three Dimensional	mass/volume	lbs / cu-in

Where D is the real system damping matrix, and α_i and β_1 are the proportional mass and damping coefficients for block i. These coefficients are completely analogous to the system level coefficients described in section 2.28. The damping contributions from these block parameters are always added to the other contributions.

Block specific damping is applied using the **blkalpha** and **blkbeta** parameters. Also see section 2.21.9 for material modal like damping.

Non-Structural Mass. An element block may define a non-structural mass (nsm) to be applied in addition to the elements' internal mass. This can be used to simulate an external load being placed on the elements. It is specified as a pseudo density, and the units depend on the type of element being used. Table 20 list these units.

The following is an example of how to use non-structural mass in the input file:

```
//nsm specified in pounds per square inch
BLOCK 3
     material 2
     tria3
     thickness 0.01
     nsm 0.005
END

MATERIAL 2
     density 0.5
END
```

2.20 Macroblock

It is possible to overload a single element block in the **Exodus** file to be used simultaneously as several different element types. To use this feature, the **BLOCK** entry

Table 21: Element Attributes

Element Type	attr	keyword	Description
ConMass	1	Mass	concentrated mass
	2	Ixx	xx moment of inertia
	3	Iyy	yy moment of inertia
	4	Izz	zz moment of inertia
	5	Ixy	xy moment of inertia
	6	Ixz	xz moment of inertia
	7	Iyz	yz moment of inertia
	8,9,10	offset	offset from node to CG
Beam	1	Area	Area of beam
	2,3,4	Orientation	orientation vector. For the orthogonal direction
	5	I1	First bending moment
	6	I2	Second bending moment
	7	J	Torsion moment
	9,10,11	offset	beam offset
Spring	1	Kx	spring constant in X
	2	Ky	spring constant in Y
	3	Kz	spring constant in Z
Triangle	1	thickness	thickness
	2	offset	shell offset in normal direction
Quad	1	thickness	thickness
	2	offset	shell offset in normal direction

should list the ids of the new macroblocks, which will share the same geometry from the **Exodus** file. Additional parameters should not be included in the **BLOCK** specification as the original element block will be treated as a "dead" element. For every macroblock listed, a **Macroblock** entry must be present in the input file. A **Macroblock** entry should look exactly like a normal **BLOCK** entry except for the keyword. The macroblock ids must be unique and different from any existing block ids.

Macroblock Example. An example is provided below.

```
// the following element block is associated with block 1 in the
// exodus file.  It specifies which macroblocks use this block.
BLOCK 1
    macroblock 11 12
END

// the following macroblocks specify the types to use for the block
MACROBLOCK 11
    dashpot
    k=1e6
    c=1e4
    cid=1
END

MACROBLOCK 12
    spring
    Kx 1e+3
    Ky 1e+3
    Kz 3e-1
END
```

Macroblocks 11 and 12 will be used as though there are two distinct element blocks in the **Exodus** file, one treated as a dashpot and the other as a spring. Because the macroblocks do not actually exist in the exodus file, element variables cannot be associated with them. However, it is still possible to obtain some element variable results from the `.rslt` file (section 2.7). Macroblock results will be specially labeled in this file because their elements' ids are not unique.

2.21 MATERIAL

Most element blocks must specify a material. Details of that material are included in the material section. The material section contains a material identifier (which is usually an integer, but may be any string), an optional **name** keyword followed by a material name, a material type keyword and the necessary parameters. The different material types and their parameters are summarized in Table 23.

For example,

```
MATERIAL 3
    isotropic
    name "steel"
    E 30e6
    nu .3
END
```

Deterministic materials may be input as **isotropic**, **orthotropic**, **orthotropic_prop**, **anisotropic**, or **isotropic_viscoelastic**. In addition, stochastic isotropic materials may be specified as **S_isotropic**.

2.21.1 Isotropic Material

Isotropic materials require specification of two of the following parameters.

Parameter	Description
E	Young's Modulus
nu	Poisson's Ratio
G	Shear Modulus
K	Bulk Modulus

Isotropic materials are the default, and the keyword **isotropic** is not required.

2.21.2 Anisotropic Material

Anisotropic materials require specification of a 21 element C_{ij} matrix corresponding to the upper triangle of the 6x6 stiffness matrix. Data is input in the order C_{11},C_{12}, C_{13}, C_{14}, C_{15}, C_{16}, C_{22}, etc. The C_{ij} must be preceded by the keyword **Cij**. The

keyword **anisotropic** is also required. Materials are specified in the order xx, yy, zz, zy, zx, xy. Note that this ordering varies in the literature. It differs from the ordering in Nastran and Abaqus, but is consistent with much of the published materials science data. An example input file with an anisotropic material is found in section A.2.

2.21.3 Orthotropic Material

Orthotropic material entry is identical to the anisotropic case with the exception that the keyword **orthotropic** replaces **anisotropic**, and only 9 C_{ij} entries are specified. These entries correspond to C_{11}, C_{12}, C_{13}, C_{22}, C_{23}, C_{33}, C_{44}, C_{55} and C_{66}. Like the anisotropic material definition, the order is xx, yy, zz, zy, zx, xy.

Alternatively, an orthotropic material may be specified using **orthotropic_prop** and the material parameters `E1, E2, E3, nu23, nu13, nu12, G23, G13`, and `G12` as shown in the following example. Note that all elastic materials must satisfy requirements that the elasticity matrix is positive definite.

```
Material honeycomb
   orthotropic_prop
   name 'aluminum honeycomb in Mpa'
   E1 = 508.7
   E2 = 7641.0
   E3 = 14750.0
   Nu12 = 1.293
   Nu23 = 0.3299
   Nu13 = 0.3367
   G12 = 115
   G23 = 2320.
   G13 = 450.
   density=0.5
END
```

A single orthotropic layer may be specified using **orthotropic_layer**. An orthotropic layer must specify 4 of the above parameters (`E1, E2, nu12, G12`). Here is an example:

```
Material 13
   orthotropic_layer
   name 'ortho layer 1'
   E1 = 508.7
```

```
      E2 = 7641.0
      Nu12 = 1.293
      G12 = 115
      density=0.5
   END
```

If sensitivity analysis is being performed (see section 2.27), one indicates the parameters for analysis by following these parameters with the `+/-` characters. In the first entry method, a sensitivity analysis must be performed on all 9 parameters. In the second, each individual parameter must be requested individually. The concept is that the sensitivity is performed with respect to the labeled parameters, i.e. either the set of C_{ij} parameters, or each individually labeled `E1` term.

2.21.4 Stochastic Material

For stochastic materials, all material properties are determined by a table look-up, based on the element ID. The file name for the table lookup is taken from the **name** identifier. The file is a standard text file with the first column corresponding to the element ID. The second column is the bulk modulus, K, and the third (and final) column is the shear modulus, G. The element IDs in the file need not be continuous, but they must be sorted in increasing order. Thus the **S_isotropic** data lookup file contains the element ID, the bulk modulus and the shear modulus, with one line for each element. The stochastic material model is very preliminary and is expected to change significantly in the next few years. An example section from the input file is presented below.

```
MATERIAL 3
        s_isotropic
        name "mat3.txt"
        density 0.288
END
```

From within "`mat3.txt`" the data looks like the following. The last two columns are bulk and shear moduli respectively.

```
1   40e6  20e6
2   40e6  20e6
4   40e6  20e6
9   40e6  20e6
10  40e6  20e6
11  40e6  20e6
```

2.21.5 Linear Viscoelastic Material

Linear viscoelastic materials require the specification of the density, and the limiting moduli E_g, E_inf, G_g, G_inf. The subscript 'g' refers to the glassy modulus, which occurs at $t = 0$, or $\omega = \infty$. The subscript 'inf' refers to the rubbery modulus, which occurs at $t = \infty$, or $\omega = 0$. In addition the Prony series for the viscoelastic materials have to be specified using keywords K_coeff, K_relax, G_coeff, and G_relax. All of these parameters are required.

For the bulk modulus K, the Prony series parameters are defined by the following equation:

$$K(t) = K_{inf} + (K_g - K_{inf}) \sum_i K_{coeff}[i] * e^{-\frac{t}{K_{relax}[i]}} \tag{22}$$

A similar equation holds for the shear modulus. Note that, the K_coeff and G_coeff MUST sum to 1.0 (individually). Otherwise, the formulation is inconsistent. That is,

$$\sum_i K_{coeff}[i] = \sum_i G_{coeff}[i] = 1.0 \tag{23}$$

Note that the number of terms in K_coeff and K_relax must be the same, and the number of terms in the G_coeff and G_relax must be the same. However, the number of terms in the K series does not have to equal the number of terms in the G series. Thus, one could simulate a case where the material shear modulus G is viscoelastic, but the bulk modulus is not. In this case, the latter would have no terms in its series.

Optional parameters for viscoelastic materials include reference (T_0) and current temperature (T_current), and the WLF constants C_1 and C_2. (more explanation of the Williams-Landel-Ferry (WLF) equation is given below). Also, two constants may be specified that describe the curve fit for the shift function, a_T1 and a_T2, in the case when T_current - T_0 is negative. The equation was provided by Terry Hinnerichs and is a good characterization of many viscoelastic materials. Its form is

$$a\_T = a\_T1 * (1 - e^{a\_T2*(T\_current - T\_0)}) \tag{24}$$

If these optional parameters are not specified, default values are used, as shown in the table below. Note that equation 24 will only be used to compute the shift functions if the parameters aT_1 and aT_2 are specified. Otherwise, the standard WLF equation is used, as described below.

If the parameters aT_1 and aT_2 are not specified, then the shift factors are computed using the WLF equation. This equation is frequently used to determine an approximate set of shift factors when experimental data for a particular material

Table 22: Default Parameters for Viscoelastic Materials

parameter	default value
T_0	0.0
T_current	0.0
C_1	15.0
C_2	35.0
aT_1	6.0
aT_2	.0614

is not at hand. The shift factors computed from this equation are used to scale the coefficients in the Prony series. The shift factors computed from the WLF equation are a strong function of temperature. The WLF equation is as follows

$$log(a_T) = -\frac{C\_1(T\_current - T\_0)}{C\_2 + T\_current - T\_0} \tag{25}$$

where T_current is the current temperature in the block, and T_0, C_1, and C_2 are material parameters that are determined experimentally. If C_1 and C_2 are not known for a particular material, then the default values given above are typically used. Typically, T_0 is the glass transition temperature of the material of interest. More explanation of the WLF equation can be found in the books by Aklonis,[9] and Ferry.[10]

After computing the shift factors using one of the two approaches given above, the relaxation times are shifted. This occurs before computations begin, using the relations

$$G_{coeff}[i] = a_T G_{coeff}[i] \tag{26}$$
$$G_{coeff}[i] = a_T G_{coeff}[i] \tag{27}$$
$$\tag{28}$$

These shifts are automatically computed given T_0, T_current, C_1, and C_2, so that the user does not need to shift the relaxation times beforehand. Note that if these parameters are not specified in the input file, then they are given default values that result in no shifting of relaxation times. In such a case, $a_T = 1$.

An example material block for a linear viscoelastic material looks like:

```
MATERIAL 9
 isotropic_viscoelastic
```

```
name "foam"
T_0=0
T_current=25
C_1=1
C_2=2
aT_1=6.0
aT_2=.06
K_g 30.0e6
K_inf 10.0e6
G_g 10.0e1
G_inf 12.0
K_coeff .5 .5
K_relax 3.0 2
G_coeff .5 .5
G_relax 1 3
density 0.288
END
```

Note that the coefficients of both K and G sum to 1.0. This is necessary for a consistent formulation.

A note on viscoelastic materials: when using viscoelastic materials in a nonlinear transient simulation, it is necessary to specify "nonlinear=no" in the BLOCK section of the viscoelastic block. This is because different internal force mechanisms are called for linear and nonlinear cases, and viscoelastic materials in Salinas only support linear constitutive model and small deformation.

We also note that if viscoelastic materials are used in a statics simulation, then the material is assigned the properties G_{inf} and K_{inf}. This is because in a slow (static) loading, the material would respond with these material properties since they are the long-time or slow response properties.

2.21.6 Acoustic Material

Linear acoustic materials require the specification of the fluid density, and the linear speed of sound. In addition, the keyword **scalaracoustic** must be in the material block.

```
MATERIAL
  name "air"
  scalaracoustic
```

```
  density 1.293
  c0 332.0
END
```

Nonlinear acoustic materials require one additional parameter, B_over_A, which is a measure of fluid nonlinearity. For air, $B\_over\_A = 0.4$. Tables of B_over_A for various fluids can be found in.[11]

2.21.7 Temperature-Dependent Material Properties

Material properties in Salinas can be specified to be temperature dependent. Temperature dependent material properties are only supported when temperatures are read in from a Calore output file. Thus, in that case, the material properties would vary from element to element, since the temperatures vary with each element. When temperatures are specified on a block-by-block basis, the temperature-dependence of the material properties can be specified explicitly in the input deck.

For linear elastic materials, an example of specifying temperature dependent properties is given below.

```
MATERIAL 1
        E 10e6
          function=1
        alphat .001
        tref 100
        nu 0.0
        density 7700.0
END

MATERIAL 2
        E 10e6
          function=2
        alphat .001
        tref 100
        nu 0.0
        density 7700.0
END

FUNCTION 1
       type LINEAR
       data 0.0 4.0
       data 5.0e9  4.0
```

```
END

FUNCTION 2
       type LINEAR
       data 0.0 3.0
       data 5.0e9  3.0
END
```

In this case, the elastic modulus of material 1 is specified by function 1, and the elastic modulus of material 2 is specified by function 2. The moduli of each element will be determined from its temperature and an interpolation on the function. In this example, the functions are trivial, and thus the moduli of materials 1 and 2 will be 4 and 3, respectively. Note that any of the 4 elastic constants k, g, e, ν can be specified as temperature dependent, and can be given different functions. In this example, the Poisson ratio is constant and only the elastic modulus is temperature dependent. Also, even though values of $10e^6$ are given for the moduli in this example, these values are overridden by the presence of the line "function=1" and "function=2".

We note that for viscoelastic materials, functions do not need to be specified in the **material** block to designate temperature dependence of the shift factors. This is accounted for automatically. See the section on viscoelastic materials for more details.

Currently, only linear elastic and linear viscoelastic materials can be given temperature-dependent material properties.

2.21.8 Density

For solutions requiring a mass matrix, all material specifications require a keyword **density** followed by a scalar *value*.

Table 23: Material Stiffness Parameters

material type	parameters
isotropic	any two of K, G, E or ν
orthotropic	nine C_{ij} entries
orthotropic_prop	$E1$, $E2$, $E3$, $nu23$, $nu13$, $nu12$, $G23$, $G13$, $G12$
anisotropic	21 C_{ij} entries
S_isotropic	file containing K and G

2.21.9 CJetaFunction

For the **CJdamp** solution method (see section 2.1.5), a frequency dependent damping coefficient, $\eta(f)$, may be specified[‡‡]. All other solution methods will ignore this keyword. The **CJetaFunction** keyword requires as a parameter the identifier of a function. Its use is specified in the following example. See section 2.23 for details in specifying the function. If no function is specified, the block will be treated as if the function were identically zero everywhere.

```
MATERIAL 1
  E=10.0E6
  NU=0.28
  density=0.098
  cjetafunction=1
END

function 1
  name 'function to use for material 1 eta'
  type linear
  data 0.0 0.001
  data 100 0.010
  data 200 0.030
  data 400 0
end
```

The function specifies the frequency, amplitude pairs for η. The frequencies are in the same units as the modal frequencies (i.e. there is no factor of 2π, and they are usually supplied in Hertz). The **CJdamp** solution process interpolates the function at the eigen frequencies to determine the effective damping for any particular mode.

2.22 COORDINATE

Coordinate systems may be defined for reference to the materials and boundary conditions. As reported in the "history" section, nodal results may also be reported in arbitrary coordinate frames in the history file only (see section 2.9). Note that all nodal locations, outputs, etc. are always defined in the basic coordinate system in the standard exodus files. These new coordinate systems are always defined based on three *locations*, which are defined in the basic coordinate system. These locations are illustrated in Figure 3.

[‡‡] η is twice the normal modal damping coefficient. Thus, if eta=0.02 for all materials, the equivalent modal damping will be 1 percent.

1. The location the origin of the new coordinate system, v_1.
2. A point on the Z axis of the new system. This is illustrated in the figure by the vector v_2. Note however, that the location is required, which is the vector sum of $v_1 + v_2$.
3. A point in the $\tilde{X}\tilde{Z}$ plane of the new system, illustrated by the vector v_3. Note that vector v_3 need not be orthogonal to v_2, but it may not be parallel to it.

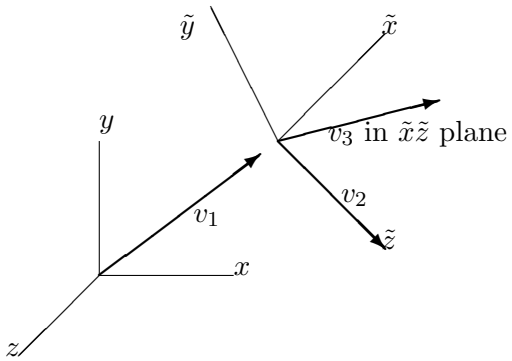


Figure 3: Coordinate System Definition Vectors

Coordinate systems for cartesian, cylindrical and spherical coordinates may be defined. In the case of noncartesian systems, the XZ plane is used for defining the origin of the θ direction only.

This example creates a cylindrical system located at a point (1,1,1) with the cylindrical axis in the (0,0,1) direction and the radial coordinate in the global Y direction.

```
Coordinate 7
    cylindrical
    1 1 1
    1 1 2
    1 2 1
END
```

The keywords for the coordinate system definitions are:

1. RECTANGULAR or CARTESIAN to define a cartesian system,
2. CYLINDRICAL for a cylindrical, i.e. polar system, and
3. SPHERICAL for a spherical system.

If "input" is selected in the ECHO section then the transformation matrix will be output in the `.rslt` file (section 2.7). The transformation matrix is a unitary matrix which can be used to transform vectors from one system to another. If we let T be the matrix reported in the `.rslt` file, then the transformation from the basic system to the rotated frame is given by,

$$v_{new} = T^T v_{basic}$$

where v_{new} is the vector in the new coordinates,
v_{basic} is the vector in the basic system, and
T^T is the transpose of the `.rslt` matrix reported.

While the history file provides a convenient means for transforming coordinates, its applicability may be somewhat limited. In particular, only a single history file is written in each analysis, and only one coordinate frame may be output per node (see section 2.9).

2.23 FUNCTION

Time or frequency dependent functions for transient and frequency response analysis can be defined using the **function** section. The following examples illustrate the use of this section.

```
FUNCTION 1
  type LINEAR
  name "test_func1"
  data 0.0 0.0
  data 0.0150 0.0
  data 0.0152 1.0
  data 0.030 0.0
END

FUNCTION 2
// This is a smooth pulse with time duration .05
//  it peaks at approximately t=.02 sec with a
//  value of 0.945.
```

```
// The equation is y(t)=-800*t^2 + 8.9943*sqrt(t)

  type POLYNOMIAL
  name "poly_fun"
  data 0. 0.
  data 2.0 -8.0e2
  data 0.5 8.9443
END
```

The keywords for the function definitions are:

1. TYPE to define the functional form,
2. NAME for reference in echo and output, and
3. DATA for the functional parameters.

Currently there are four types of functional forms, **linear**, **table**, **polynomial** and **loglog**. The data elements are defined in the following paragraphs.

2.23.1 Linear Functions

For linear functions, the data elements are points of the function where the user defines the value of the independent variable (e.g. time) and the corresponding value of the function. Linear interpolation is used to find all other values of the function. In order to make the linear interpolation unique, the order of the input data is important. Input checks will ensure that time on subsequent data points is always greater than or equal to time on the previous data point so that curves cannot double back on themselves. For example,

```
FUNCTION 3
    name "illegal_fun"
    type linear
    data 0.00 0.
    data 0.01 1.
    data 0.05 1.
    data 0.04 0.   //illegal. the first column must never decrease
END
```

Linear functions will extrapolate by using the value of the nearest data point. For example, in the following function, f(t=0.3) = 0.5.

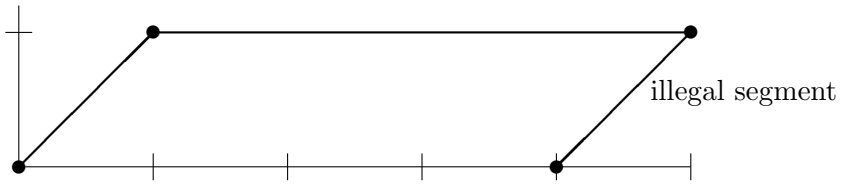


Figure 4: Linear function #3. "illegal_fun"

```
FUNCTION 5
    name "extrap_fun"
    type linear
    data 0.00 0.
    data 0.01 1.
    data 0.02 0.5
END
```

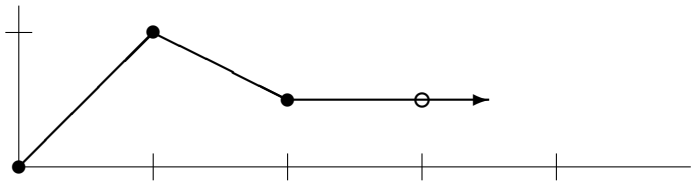


Figure 5: Linear function #5. "extrap_fun"

2.23.2 Functions using Tables

Functions may be specified by reference to a linearly interpolated **table** (as discussed in section 2.25). The table must be of dimension=1. Tables are very similar to the linear functions described above with several important differences.

1. Referencing a value of a table beyond the valid range is an error. This is seldom a problem in frequency domain analysis, but could often be an issue for time domain analysis.

2. Tables can be more memory efficient that linear functions in some cases where there is a large amount of data. This is especially important if only a few processors need access to that data.

The function in the following example is a tabular representation of the data of Figure 5 and Function 5 above.

```
FUNCTION 7
  type table
  tablename=example7
END

TABLE example7
  dimension=1
  size=5
  datafile='example7.txt'
  origin 0.0
  delta  .01
END
```

Within the datafile, "example7.txt", the following data would be represented.

```
0.0
1.0
0.5
0.5
0.5
```

Of course, the linear function can be evaluated for any time, and the table is limited to the range 0-0.04. Table type functions require the **tablename** keyword.

2.23.3 Polynomial Functions

For polynomials, the data points given are the exponent of the independent variable and a scale factor for that term. The independent variable taken to any real power will always be evaluated as positive. If powers are repeated, their coefficients will sum. For example,

```
FUNCTION 6
    name "poly_fun"
    type polynomial
    data 0.0 0.
    data 1.0 1.
    data 2.0 0.1
    data 1.0 0.5
END
```

is equivalent to

```
FUNCTION 6
    name "poly_fun"
    type polynomial
    data 0.0 0.
    data 1.0 1.5
    data 2.0 0.1
END
```

The function value as a function of the independent variable t is,

$$f(t) = 1.5t + 0.1t^2.$$

2.23.4 LogLog Functions

In frequency domain analysis, log/log functions are commonly used for application of loads. This is particularly true for random vibration inputs which are commonly specified on log/log plots. The **loglog** option allows linear interpolation on a log/log plot so that only the corner frequencies need be specified. An example follows.

```
FUNCTION 1
   name "my_loglog"
   type loglog
   data 1.0 1e-8
   data 299 1e-8
   data 300 0.01
   data 2000 0.03
   data 8000 0.03
   data 10000 0.01
   data 10001 1e-8
END
```

2.23.5 Random Functions

There are two different types of a random function distribution: a uniform and a Gaussian distribution. For both distribution types, the values are randomly generated according to the range that is input.

For uniform distributions, the left range number is the lower bound and the right number is the upper bound, both inclusive. For a Gaussian distribution, the

left number is the mean (or center of the distribution), and the right number is the standard deviation.

The *seed* determines the seed for a new sequence of pseudo-random numbers. There are two options, auto or a positive integer number. With auto, the computer clock is used to seed the generator, which will nearly always give an unpredicatable string of random numbers. However for repeatable results, a manual seed may be given. The sequence of numbers is random, but the same random sequence of numbers generated from a specific seed is always the same. Please note that the number 0 acts the same as if you had entered auto as the seed.

Random functions use the pseudo-random number generator in the rand() fucntion of the C library.

```
FUNCTION 2
      name "some_function"
      type random
      distribution gaussian
      range -1.0 4.0
      seed auto
END
```

That example would produce the distribution shown in figure 6:
Parameters are shown in Table 24:

Table 24: Random function parameters

Parameter	Type	Values
distribution	string	*uniform* or *gaussian*
range	two Real numbers	lower and upper bound of distribution (uniform) OR mean and standard deviation (gaussian)
seed	string/integer	*auto* OR any integer

2.23.6 User Defined Functions

A user defined function capability has been added to Salinas to permit application of generic functions that cannot be readily evaluated using built in functions. Note the following.

1. User defined functions are typically quite slow.
2. By default, user defined functions are evaluated at each application point on the structure (i.e. each node in a nodeset). Thus, they must be evaluated

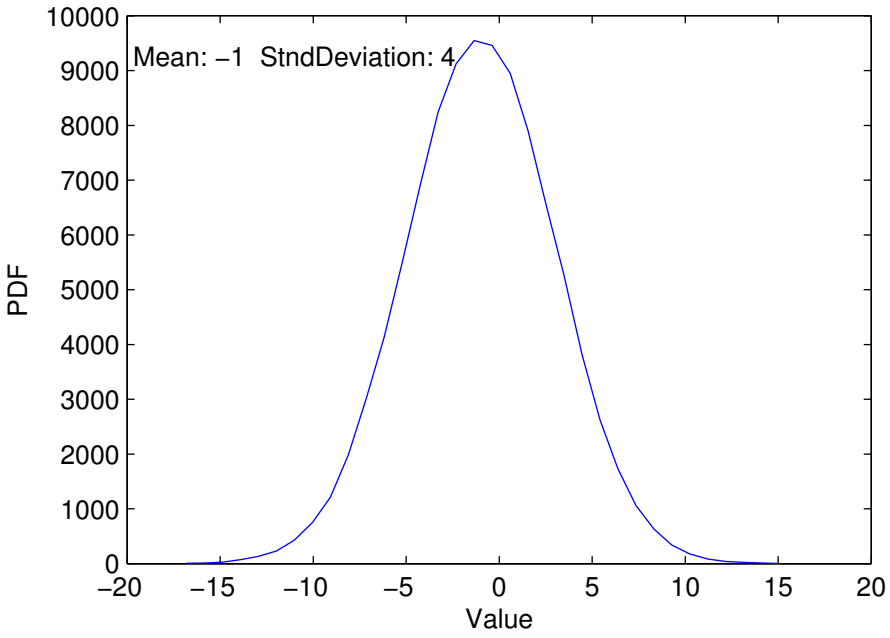


Figure 6: Example *Gaussian* output.

many times. This with their slow evaluation can result in significant time for their evaluation. If you can do the problem another way, it is strongly recommended that you do so.

3. User defined functions are impossible to fully test in our test environment.

4. User defined functions may be less robust than other methods.

Salinas uses the runtime compiler (RTC) environment that was developed for Alegra.[12] This environment was chosen for several reasons, but the primary goals are to provide capabilities that cannot be performed in other ways, and to do with a simple, portable system.

The Alegra RTC is a library that compiles a subset of the "C" language in run time. The user is referred to the RTC documentation for details of the library. The RTC function is referenced in the salinas input just as any other function. For example,

```
LOADS
   nodeset 1
      function=1
      force=0 0 1
END

FUNCTION 1
   type=USER
   rtcfile='example.cc'
END
```

The function "type" is defined as "USER". In addition, the "rtcfile" parameter must be specified. The rtcfile points to the file containing the source for the function. Typically the file name has a ".cc" extension (to indicate that it is C++ source), but any filename is acceptable, and either a relative or a full path may be specified.

The permitted parameters are listed in the following table.

Parameter	**Argument**	**Description**
rtcfile	*string*	the file name containing source. *Required!*
timeonly	none	flag. If this exists, no spatial dependence is allowed in the function.

The Source File: The **rtcfile** points to a file containing source code to be compiled. This is a subset of the "C" language. There are some idiosyncracies which we list here.

Variable	I/O	Description
time	input	the current time for the function evaluation.
retvar	output	The function return value
coord	input	The undeformed coordinates of the node
disp	input	The deformation vector. Dimension=7
velocity	input	The velocity. Dimension=7
acceleration	input	The acceleration on this node.

Table 25: Predefined RTC variables

- No comment fields are allowed.
- No function definitions are allowed.
- Data is passed to and from Salinas using specifically named, predefined variables. These are listed in Table 25.

We provide an example below for the case of a force that is inversely proportional to the deformed Z coordinate of each node in a sideset. This distance is labelled R in the script, it is checked for divide by zero, a function value is computed, and the value returned in the *retvar* variable. This function will be run on all the nodes in a side set.*

```
double R=disp[2]+coord[2];
retval=1e10;
if ( abs(R) > 1e-10 ){
   retval = 1.0 / R;
}
```

2.24 MATRIX-FUNCTION

This section provides for input of a matrix function as is used in a cross correlation matrix for input to a random vibration analysis. In the limit of a single input these reduce to a single function (as described in the previous section). Note that a matrix-function can have arbitrary symmetry and can be complex. An important feature

* The HowTo document has an example of an RTC analysis together with suggestions on debugging.

of the matrix-function is that each entry of the matrix is a function of frequency (or time).

The Matrix-Function is illustrated in the following example.

```
MATRIX-FUNCTION 1
   name 'cross-spectral density'
   symmetry=hermitian
   dimension=2x2
   nominalt=20.1
   data 1,1
     real function 1 scale 1.0
   data 1,2
     real function 12
     imag function 121 scale -3.0
   data 2,2
     real function 22 scale 0.5
 END
```

Matrix functions have the following parameters.

NAME allows you to optionally enter a string by which the matrix-function will be identified in subsequent messages.

SYMMETRY identifies the matrix symmetry. Options are "none", "symmetric", "asymmetric" and "hermitian". If the matrix is not square, only "none" can apply. The default for this optional parameter is "symmetry=none".

DIMENSION specifies the dimension of the matrix. If not specified, it defaults to 1x1. The dimension is specified as the number of rows, an "x" and the number of columns. No space should be entered between the terms.

DATA Each data entry specifies one entry in the matrix-function. The *data* entry must be immediately followed by the matrix location specified as a row, column pair. Again, no spaces may be inserted in the location entry. The *data* parameters uses two keywords.

- "real" identifies the real component of the entry. It must be followed by a function reference (see section 2.23), and optionally by a scale factor.
- "imag" identifies the imaginary component of the entry. It must be followed by a function definition, and an optional scale factor.

NOMINALT Used only for echoing the matrix values. If *input* is specified as an *Echo* option (see section 2.7) general information from the matrix function are written to the log file (the *.rslt* file). If, a *nominalt* entry also exists, then the matrix entries are written for that nominal time (or frequency). Only one such output can be specified. It provides a means of checking the input to assure the matrix values are correct at a single time (or frequency) value.

2.25 Table

The **Table** section permits construction of tabular data in 1 to 4 dimensions. Tables must be referenced in other structures for their data to be useful. Tables are characterized by data structures which are sampled at uniform intervals. Tables offer the following benefits.

- They provide implicit linear interpolants for values between tables.
- They are fairly flexible structures which are more memory friendly than functions (for some applications).
- Tables are the only way to introduce multi-dimensional data.

Each Table includes a number of required and optional parameters, as shown below.

Table 26: TABLE Section Options

Parameter	Default	Description
dimension	*required*	number of dimensions in the table
size	*required*	table size in each direction
datafile	*required*	ASCI file containing the values at each point
origin	zero	origin of the table (for scaling)
delta	1	interval between points in each direction

The **dimension** identifies the shape of the table. For example, `dimension=2` indicates a table of xy values. All other quantities depend on this dimension.

The **size** parameters indicate the individual hypercube dimensions of the table. For example, in a table of `dimension=2`, the **size** parameter indicates the number of rows and columns in the table. The total number of entries is the product of all the terms in the size.

The **datafile** parameter contains the name of a text file containing all the data values in the table. The values are entered with the first dimension cycling faster. Thus, in a `dimension=2` table, all the entries for column 1 are first entered, followed

by column 2, etc. The layout of the file is not important. Data values are read one at a time as they are separated by white space. There must be exactly the correct number entries in the file. No comments are permitted in the **datafile**.

> *Note: We recommend that the values in the table be entered as a single entry per line. It is very confusing to enter multiple entries in the table on a single line. The first row is interpreted as the entries in the first column, and so on. This leads to the table being interpreted as the transpose of the entered data.*

Both the **origin** and the **delta** parameters are optional values provided for interpolation. The implicit integer entries of the table are converted to real values for function evaluation by use of these parameters.

Function evaluations within the range of the table can be linearly interpolated. The range in each direction is determined by the following.

$$\text{origin}_i < \text{range}_i < \text{origin}_i + (\text{delta}_i \cdot \text{size}_i) \tag{29}$$

Evaluations of the table for regions outside the valid range result in a warning message.

In contrast to a **function** (see section 2.23), tables require memory only as needed. All processors store the full input file in memory. However, tables can store a large amount of data in the **datafile**. This file is opened and data is read from it only as needed. For this reason, tables are preferred over functions when only a few processors may need access to a large amount of data. Obviously, tables are the only option when a function of more than one variable is required.

An example of a two dimensional table definition is shown below.

```
Table example-2d-table
   dimension=2
   size    = 200 300    // note: don't put in an x
   origin    1.0 0.0    // optional. defaults to 0 0
   delta     1.0 0.9    // optional. defaults to 1 1
   datafile 'junk.txt'
 END
```

2.26 CBModel

The **CBModel** section provides a method of specifying information related to a Craig-Bampton model reduction of the entire structure. It is required by the CBR method (section 2.1.7).

The "interface" is that portion of the model which will interface to the external structure. The interface is defined by collections of nodes specified as nodesets or sidesets. After eliminating boundary conditions, the active degrees of freedom on the nodes become the interface.

Table 27: CBModel Parameters

Keyword	**type**	**Description**
nodeset	*integer*	exodus nodeset. Must include the nodeset id.
sideset	*integer*	exodus sideset. Must include the sideset id.
format	*string*	specifies the output format. **m**atlab - matlab .m format **d**mig - nastran DMIG format **ne**tcdf - netcdf format†
file	*string*	specifies the file name for output.
output_vector	*string*	'yes' to output the constraint modes and fixed interface normal modes
GlobalSolution	*string*	'yes' to compute the eigen solution of the reduced system.

†The netcdf format is the database upon which exodusII is built. A translator from this format to nastran `output4` format is available.

The supported keywords for the CBMODEL section are shown in Table 27. The keywords are described below.

nodeset: The **nodeset** keyword specifies the nodes to be placed in the interface. Nodesets are defined in the exodus file. An integer nodeset ID must follow the nodeset keyword. Alternatively, a list of nodesets (in matlab type format) can be specified. This is identical to the `history` file definition of section 2.9.

sideset: A **sideset** may also be used to specify the interface nodes. Any number of nodeset and sideset combinations are allowed. The interface is the union of all such entries.

format: The preferred format is the **netcdf** format. This is actually a superset of the exodus format. It is the format that must be used if the reduced model is to be inserted into another **Salinas** model as a superelement. The **dmig** format is for use with nastran, and will probably be dropped in the future. It contains only the reduced system matrices (no maps, coordinates, etc). The matlab format is a convenience.

Note that the **netcdf** format may be converted to the other forms using a stand alone translator, **ncdfout**.

file: The **file** keyword is required to specify the output file name.

output_vector: Specification of the **output_vector** provides output of the constraint modes and fixed interface modes used to compute the reduced order system.

GlobalSolution: As a convenience, we will optionally compute the eigen values of the reduced system. It is strongly recommended that these values be compared with the eigenvalues of the full system to insure that the model has converged over the frequency of interest.

Data in Table 28 will be written to a file. The Output Transfer Matrix (or OTM) depends on data in the **History** section (see section 2.9). Specifically, the output nodes and elements, and the output variables are specified in the history file *as if they were to be output to a history file.* For simplicity, and because the OTM describes a linear transfer matrix, only a limited subset of results are provided. In particular, displacements and the natural strains and stresses may be written. The transfer matrix provides the following computation.

$$\begin{bmatrix} u \\ \epsilon \\ \sigma \end{bmatrix}_{out} = \begin{bmatrix} \text{OTM} \end{bmatrix} \begin{bmatrix} q \\ u_{int} \end{bmatrix}$$
$$= \begin{bmatrix} \Phi_u & \Psi_u \\ \Phi_\epsilon & \Psi_\epsilon \\ \Phi_\sigma & \Psi_\sigma \end{bmatrix} \begin{bmatrix} q \\ u_{int} \end{bmatrix}$$

Here q is the amplitude of the internal constraint modes (typically computed in the next level analysis), and u_{int} is a vector of interface displacements. The fixed interface modes (eigen modes of a clamped boundary) are represented by Φ, and the constraint modes by Ψ.

The left hand side vectors represents internal results (displacement, strain and stress) which are computed from the interface results. Any of the output results may be omitted, and the OTM will retain only nonzero components. For example, if only displacements are required, the matrix reduces to $[\Phi_u \ \Psi_u]$. The OTM matrix is a rectangular matrix, and it is typically full. An example **CBModel** section follows.

```
CBMODEL
```

Table 28: Data output for Craig-Bampton Reduction

<table>
<tr><th>Variable</th><th>Description</th></tr>
<tr><td>NumC</td><td>number of constraint modes</td></tr>
<tr><td>NumEig</td><td>number of fixed interface modes</td></tr>
<tr><td>Kr</td><td>Reduced stiffness matrix.</td></tr>
<tr><td>Mr</td><td>Reduced mass matrix.</td></tr>
<tr><td>cbmap</td><td>A two column list providing a map from each interface degrees of freedom to the node and coordinate direction of the global model.
The first column of this list is the node number (1:N) in the structure. The second column indicates the coordinate direction as follows.
<table><tr><th>Number</th><th>Description</th></tr><tr><td>1</td><td>x</td></tr><tr><td>2</td><td>y</td></tr><tr><td>3</td><td>z</td></tr><tr><td>4</td><td>Rotation x</td></tr><tr><td>5</td><td>Rotation y</td></tr><tr><td>6</td><td>Rotation z</td></tr><tr><td>7</td><td>acoustic pressure</td></tr></table>The “cbmap” has the same number of rows as Kr or Mr.</td></tr>
<tr><td>OutMap</td><td>A map of the nodes in the output transfer matrix. OutMap(i) is the global node number for each node in the output. There are always 6 rows of output for each node. Thus OutMap(1) corresponds to rows 1 through 6 in the OTM.</td></tr>
<tr><td>OTM</td><td>Output Transfer Matrix to provide a transfer function from the interface dofs to internal degrees of freedom or other results.</td></tr>
<tr><td>OutElemMap</td><td>A map of the elements in the output transfer matrix, OTME. OutElemMap(i) is the global element number for each element in the output. There are always 6 rows of output for each element.</td></tr>
<tr><td>OTME</td><td>Output Transfer Matrix to provide a transfer function from the interface dofs to internal elements.</td></tr>
</table>

```
    nodeset=1:2          // nodes from nodeset 1 and 2
    format=netcdf        // use a netcdf format file
    file='junk.ncf'
END
```

NOTE:
In release 2.2 we released the OTM output capability. This permits an analyst to output the reduced order model of the entire structure for use in another code that supports superelements (such as MSC/Nastran). In a subsequent release, we will add the capability to input these matrices as a superelement in Salinas. At that point one could perform a Craig-Bampton reduction to generate a reduced order model of that portion of the structure. A follow up analysis could use this as a superelement. See details in Figure 7.

2.27 SENSITIVITY

This section controls global parameters related to sensitivity analysis. Sensitivity analysis is not performed in **Salinas** unless this section is present in the input file. The following example illustrates the legal keywords.

```
SENSITIVITY
  values all
  vectors 1 thru 3  5  7 thru 9
  iterations 8
  tolerance 1e-7
END
```

The keywords **values** and **vectors** are used to control what types of sensitivities are computed for which cases in the analysis. In modal analysis, these refer to the eigenvalues and eigenvectors, respectively, and the case numbers represent the mode numbers. In static and transient analysis, **vectors** refers to the displacement vector results, and **values** has no meaning. Also, in modal analysis, eigenvalue sensitivities are always computed when eigenvector sensitivities are requested for a mode. Allowable values are:

```
vectors all              // compute for all cases/modes
vectors none             // compute for no cases/modes
vectors                  // default, same as all
vectors 1 2 3 5          // cases/modes 1,2,3,5
vectors 1 thru 3 5       // using thru to define range
```

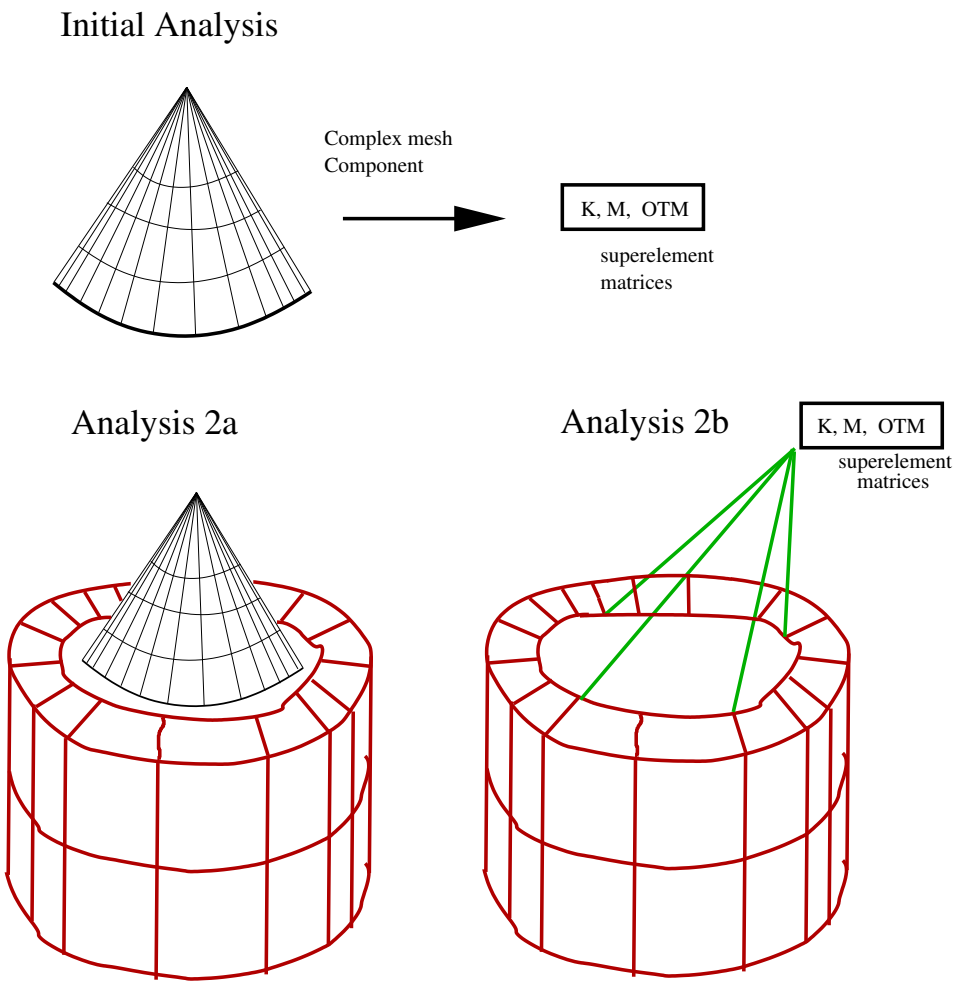


Figure 7: An initial analysis using CBR can be applied to reduce a complex component to much smaller matrices. In subsequent analyses the superelement replaces the complex component in the system analysis. There is little loss of accuracy, but significant computational benefit.

Omitting the keyword **vectors** (or **values**) is equivalent to not requesting those sensitivities; in other words, it is equivalent to **vectors none**. The keywords **iterations** and **tolerance** are used in computing eigenvector derivatives. The default values are `10` and `1.0e-06`, respectively.

Sensitivity results are scaled by multiplying the derivative with respect to a parameter by the nominal value of that parameter. In this was, the units of the sensitivity coefficients are the same as the units of the nominal response results. Furthermore, in order to determine the absolute change in a response resulting from a relative change in a parameter, simply multiply the sensitivity of the response with respect to that parameter by the relative change. For example, multiply by 0.10 for the effect of a 10% change in the parameter.

Sensitivity results are output to the same file as the nominal results. The arrangement of the output varies depending on the analysis. For statics, the nominal result is output, followed by the sensitivity result for each parameter. For eigenanalysis, the nominal frequencies and eigenvectors are output, followed by the eigenvalue and eigenvector sensitivities with respect to the first parameter, the second parameter, and so on. The eigenvalue sensitivities are placed in the time field of each output record, just as the frequencies are for the nominal modal parameters. For transient analysis, the nominal response for each time step is output, followed by the sensitivities for that time step. Then the nominal results for the next time step are output, and so on.

The selection of parameters is controlled by the inclusion of a +/- symbol following a parameter in the input deck. Examples of valid sensitivity parameter definitions are:

```
MATERIAL 1
  E 10e6 +/- 1e6                // absolute tolerance specified
  density 2.59e-4 +/-           // no tolerance, use default
END

BLOCK 1
  area 0.10 +/- 5 %             // relative tolerance specified
END

BLOCK 2
  thickness  +/- 1 %            // relative to exodus attr
END

LOADS
```

```
  nodeset 1
  force 0. 0. 1000 +/- 0 0 10  // tolerance for vector param
END
```

Note that the tolerances are specified on the parameters where they normally appear in the input file. That is, these definitions do not appear in the **SENSITIVITY** section.

2.28 DAMPING

This section allows input of simple global viscous damping models, using either modal damping rates or stiffness and mass proportional damping. The various options for the DAMPING section are shown in Table 29.

Table 29: DAMPING Section Options

Parameter	Description
alpha	mass proportional damping parameter (real)
beta	stiffness proportional damping parameter (real)
gamma	uniform modal damping rate (fraction of critical) (real)
mode	individual modal damping ratio (fraction of critical) (integer, real)
ratiofun	index of function to define modal damping ratios

The damping matrix or modal damping coefficient is determined by summing contributions from all damping parameters given in Table 29. For modal superposition-based analysis, including **modalfrf**, **modalranvib** and **modaltransient**, all the given parameters are defined. For direct implicit transient analysis, the modal damping parameters apply only to modes for which eigenvalues and eigenvectors have previously been computed. This depends on the presence of the keyword **nmodes** in the solution section of the input file.

The effect of the mass and stiffness proportional parameters on modal damping depends on the frequencies of the modes. For modal-based analysis, the damping rate for mode i with radial frequency ω_i is given as

$$\zeta_i = \alpha/(2\omega_i) + \beta \cdot \omega_i/2 + \Gamma + \texttt{mode[i]} + ratiofun(i)$$

where the viscous damping term in the modal equilibrium equation is $2\zeta_i\omega_i$. For example the following damping input section could be used in a modal transient analysis [†].

```
DAMPING
  alpha  0.001    //
  beta 0.00005    //  C = .001 * M + .00005 * K
  gamma 0.005     // 0.5 % critical
  mode  1  0.01   // 1 % of critical
  mode  2  0.005  // 0.5 % critical
```

[†]Block specific proportional damping is also available. See section 2.19.2.

```
  mode  3  0.015  // 1.5 % critical
END
```

It produces the following damping ratios.

Mode	modal damping ratio	modal viscous damping term
1	$0.015 + 0.001/(2\omega_1) + 0.00005\omega_1/2$	$0.030\omega_1 + 0.001 + 0.00005\omega_1^2$
2	$0.010 + 0.001/(2\omega_2) + 0.00005\omega_2/2$	$0.020\omega_2 + 0.001 + 0.00005\omega_2^2$
3	$0.020 + 0.001/(2\omega_3) + 0.00005\omega_3/2$	$0.040\omega_3 + 0.001 + 0.00005\omega_3^2$

In direct (i.e. non-modal-based) transient analysis, the same damping input section would produce the same damping ratios if all the modes used in the modal transient analysis were also available for the direct transient. Conversely, if no modes were available, the above damping input section would produce a physical damping matrix $C = 0.001M + 0.00005K$.

The **ratiofun** keyword permits definition of modal damping terms based on a frequency dependent function. The associated function definition (see section 2.23) provides a table lookup for damping ratios. For example, consider a system with modes at 200 and 500 Hz. The following example will establish modal damping ratios of .03 and .06 respectively. The function describes a line defined by $ratio(f) = 0.01 + 0.1/1000f$.

```
DAMPING
   ratiofun=100
END

FUNCTION 100
   type=linear
   data 0 0.01
   data 1000 0.11
END
```

2.28.1 Nonlinear transient solutions with damping

Using the stiffness proportional damping parameter **beta** in a nltransient will generate damping terms using the tangent stiffness matrix if the element is in a nonlinear block. Otherwise, the linear element stiffness matrix is used to accumulate damping terms related to the parameter **beta**.

2.29 EXTERIOR

This section allows the user to specify an exterior boundary for coupled structural acoustic simulations. Once specified, first-order non-reflecting boundary conditions are applied on this surface. The boundary is specified with a sideset.

```
EXTERIOR
  sideset=1
END
```

In the above example, sideset 1 is used to denote the exterior boundary.

Note that the sideset used to define the exterior boundary can only be attached to acoustic elements.

2.30 NOX

This section allows input of nonlinear solver options to the NOX nonlinear solver. Currently, only a small subset of the many options available in NOX and described at

`http://software.sandia.gov/nox`

are able to be specified. These include the following:

Table 30: NOX Nonlinear Solver Options

Option	Choices
nonlinear_solver_method	trust_region_based*
	line_search_based (default)
nonlinear_direction_method	newton (default)
	modified_newton
	steepest_descent
	nonlinearcg
nonlinear_linesearch_method	full_step (default)
	polynomial
	more_thunte
	backtrack
	nonlinearcg

*Default options for this choice are used. Specification of direction and linesearch methods apply only to line_search_based.

2.31 LOCA

This section allows the user to specify continuation options to the LOCA continuation package. LOCA has numerous options which are fully described at

`http://software.sandia.gov/nox`

and the majority of these options can be specified in this section. Here we merely list the available options and provide a brief description of the most important ones.

The required **continuation_parameter** keyword gives the name of the continuation parameter which is a string, possibly followed by two integers. Valid parameter names are:

Name	ID	Index
lambda	–	–
load	load ID	load component (x,y,z)
moment	load ID	moment component (x,y,z)
element_attribute	element block ID	attribute index

Parameter **lambda** is a scale factor for the total external force on the system, i.e., $r(u, \lambda) = p - \lambda f$ and is equivalent to the load stepping parameter in **NL-Statics**. Parameters **load** and **moment** represent individual applied loads or moments. The first integer argument is the load ID, and the second is the load or moment component (x, y, or z), both counting from zero (i.e., the first load in the first load block would be load ID 0, and the y component would be index 1). Parameter **element_attribute** represents an element attribute, with the ID giving the element block ID and the index giving the index of the attribute in the element attribute array. Currently, material properties are not fully supported, nor are parameter dependent boundary conditions. Hopefully in the future a simpler system for specifying the continuation parameter will be implemented.

The **bifurcation_parameter** keyword is of the same form, and is required if the **bifurcation_method** parameter in the **LOCA** block is anything other than **none**.

The keyword **continuation_restart_file** provides a file name for an Exodus file from which an initial solution can be read. After each successful step in a continuation run, the solution components (displacements and rotations) are saved in the output Exodus file with time-step label given by the current continuation parameter value. One can then start a new continuation run using an initial solution given by one of these intermediate continuation steps by supplying the previous output file as the restart file. This is often used when looking for bifurcations in a continuation run: one discovers a bifurcation has occurred at some intermediate

Table 31: LOCA Continuation Options

Option	Choices
continuation_parameter	See below (**Required**)
continuation_restart_file	any Exodus file name (No default)
continuation_restart_index	any positive integer (default 1)
continuation_method	natural
	arc_length (default)
initial_value	any real (default 0.0)
max_value	any real (default 1.0)
min_value	any real (default -1.0)
max_steps	any nonnegative integer (default 200)
enable_arc_length_scaling	0,1 (default)
goal_arc_length_parameter_contribution	any real between 0,1 (default 0.5)
max_arc_length_parameter_contribution	any real between 0,1 (default 0.7)
initial_scale_factor	any positive real (default 1.0)
min_scale_factor	any positive real (default 1.0e-8)
enable_tangent_factor_step_size_scaling	0,1 (default)
min_tangent_factor	any real between 0,1 (default 0.98)
tangent_factor_exponent	any positive real (default 1.0)
branch_switch	0 (default), 1
compute_eigenvalues	0 (default), 1
block_size	any positive integer (default 1.0)
arnoldi_size	any positive integer (default 100)
num_eigenvalues	any positive integer (default 3)
eigenvalue_tolerance	any positive real (default 1.0e-8)
convergence_check	any positive integer (default 1)
restarts	any positive integer (default 2)
frequency	any positive integer (default 1)
debug_level	any nonnegative integer (default 0)
sorting_order	lm (default), lr, li, sm, sr, si
save_eigenvalues	any positive integer (default 0)

Table 32: LOCA Continuation Options Continued

Option	Choices
bifurcation_parameter	See below (**Required for Bifurcations**)
eigenvector_restart_method	solve_dfdp (default)
	file
eigenvector_restart_file	any Exodus file name (No default)
eigenvector_restart_index	any positive integer (default 1)
bifurcation_method	none (default)
	turning_point
	modified_turning_point
	nic_day_modified_turning_point
	pitchfork
	hopf
bifurcation_parameter_initial_value	any real (no default)
perturb_initial_solution	0 (default), 1
bifurcation_perturbation_size	any positive real (default 1.0e-3)
step_size_control_method	constant
	adaptive (default)
initial_step_size	any real (default 0.1)
min_step_size	any positive real (default 1.0e-5)
max_step_size	any positive real (default 1.0)
aggressiveness	any positive real (default 0.5)
failed_step_reduction_factor	any real between 0,1 (default 0.7)
successful_step_increase_factor	any real ¿ 1 (default 1.26)
predictor_method	constant_predictor
	tangent_predictor
	secant_predictor (default)
	random_predictor
random_perturbation_size	any positive real (default 1.0e-3)
first_step_predictor_method	same as predictor (default constant)
first_step_random_perturbation_size	any positive real (default 1.0e-3)
last_step_predictor_method	same as predictor (default constant)
last_step_random_perturbation_size	any positive real (default 1.0e-3)
output_precision	any positive integer (default 3)

point along a continuation curve (e.g., the curve goes around a turning point) and then restarts the continuation near the bifurcation and tracks the bifurcation in a second parameter. There is no default for this keyword. The index (or time step) into the Exodus restart file is specified with the **continuation_restart_index** keyword and defaults to 1.

Turning point and pitchfork bifurcations require an initial guess for the null vector (eigenvector corresponding to zero eigenvalue) of the system. How this vector is computed is controlled by the **eigenvector_restart_method** keyword, which currently can be either **solve_dfdp** or **file**. For **solve_dfdp**, the initial null vector v is computed as the solution to $Kv = \frac{\partial r}{\partial \lambda}$. If the stiffness matrix K is nearly singular, then v will have a large component in the direction of the null vector. For **file**, the initial null vector is read from an Exodus file, given by **eigenvector_restart_file**. The index is given by **eigenvector_restart_index** and defaults to 1.

Salinas has a basic branch switching capability, and it is primarily intended for pitchfork bifurcations. Branch switching is enabled by setting **branch_switch** to 1. One must then supply an approximation to the tangent vector to the new branch. For pitchfork bifurcations, this tangent vector is equal to the null vector at the pitchfork point, which then can be read from an Exodus file specified by the **eigenvector_restart_file** keyword.

3 Element Library

Short descriptions of each of the types of elements follow. Most of the parameters for the element are supplied either in the database file (i.e. **Exodus** file) or in the text input file (*.inp). If parameters exist in both locations, the values specified in the text input will over ride the exodus database specification.

3.1 Hex8

The **Hex8** is a standard 8 node hexahedral element with three degrees of freedom per node. The **Hex8** element has 8 integration points. The shape functions are trilinear. It supports isotropic and anisotropic materials.

There are three variations of **Hex8**. The default element is a bubble hex element, which can be specified by **Hex8b**, or by no specification at all. The bubble element still has 8 nodes and 3 degrees of freedom per node, and thus from a user's perspective it is no different than the standard **Hex8**. The **Hex8b** element uses bubble functions,[13],[1415] to augment the standard element shape functions. It gives much better performance in bending than does the standard hex8.

The **Hex8u** specifies an under integrated Hex with properties similar to those of most commercial finite element codes. The underintegration produces an element that is soft relative to a fully integrated element. It may be specified by **Hex8** or by **Hex8u**.

The fully integrated Hex is specified by **Hex8F**. While it performs adequately when the element shape is nearly cubic, it performs quite poorly for larger aspect ratios. For most problems involving bending the Hex8u is recommended.

3.2 Hex20

The 20 node variety of Hex element provides quadratic shape functions. It is a far better element than the Hex8, and should be used if possible. The Hex20 element in Salinas is very similar to elements found in most commercial codes.

3.3 Wedge6

The Wedge6 is a compatibility element for the **Hex8**, it is not recommended that the entire mesh be built of **Wedge6** elements. They are primarily intended for applications where triangles are naturally generated in mesh generation.

3.4 Wedge15

The Wedge15 element adds midside nodes to the Wedge6. Like the Hex20 and Tet10, it has quadratic shape functions, and is a very good element.

3.5 Tet4

This is a standard 4 node tetrahedral element with three degrees of freedom per node. The **Tet4** element has one integration point. The shape functions are linear. It is not recommended to use only Tet4 elements for the entire mesh because standard, linear tetrahedra are typically much too stiff for structural applications. The **Tet4** is provided primarily for those applications where a mesh may be partially filled with these elements. If a model is constructed of all tetrahedral elements (as by an automatic mesh generator), the **Tet10** is strongly recommended over the **Tet4**.

3.6 Tet10

This is a standard 10 node tetrahedral element with three degrees of freedom per node. The **Tet10** uses 4-point integration for the stiffness matrix and 16-point integration for the mass matrix. The shape functions are quadratic. This is a very good element for use in most structural analyses.

3.7 QuadT

The **QuadT** is a 4-node quadrilateral shell with membrane and bending stiffness. The element properties and element stiffness and mass matrices are developed by internally generated **Tria3** elements, as illustrated in Figure 8. The quadrilateral is split along the shortest diagonal. It is not an optimal element, but is adequate for most applications. A more optimal element is currently under development. See the description of the **Tria3** for details on the element.

3.8 Quad8T

The **Quad8T** is an 8-node quadrilateral shell with membrane and bending stiffness. The element properties and element stiffness and mass matrices are developed by internally generated **Tria3** elements (see Figure 9. It is not an optimal element, but is adequate for most applications. Shape functions are NOT quadratic. It is compatible with the **Tria6** element, as well as with other elements based on the **Tria3**. See the description of the **Tria3** for details on the element.

Figure 8: QuadT Element
The element is generated by internally combining two Tria3 elements.

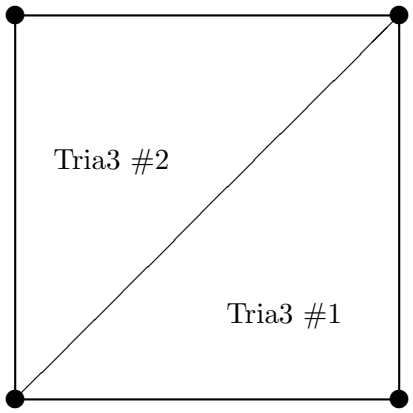
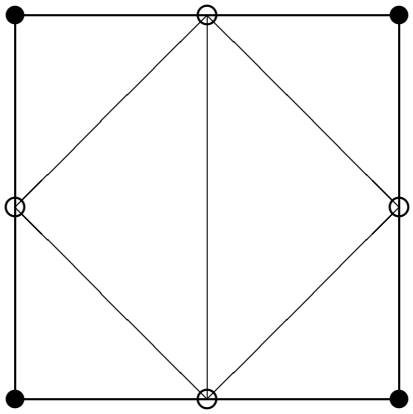


Figure 9: Quad8T Element
The element is generated by internally combining six Tria3 elements.



3.9 TriaShell

The **TriaShell** element has 3 nodes with 6 degrees of freedom (DOF) per node. The **TriaShell** is generated by decoupling the membrane DOF and the bending DOF. Allman's Triangular (AT) element[16] models the membrane DOF, while the Discrete Kirchoff Triangle[17] (DKT) models the bending DOF. These two elements are combined into the **TriaShell** element. The single layer shell supports only isotropic materials. The **TriaShell**, like the **Tria3**, has a single required attribute, `thickness`.

Additional attributes include two rotational parameters. The first is a rotation about a given axis, and the second is a rotation about the surface normal. The angles are specified in degrees and the axis is an integer 1, 2, or 3, representing the x, y, and z coordinate axes. The example below illustrates the use of these parameters.

You may also specify layers for a **TriaShell** element. When using layers, the available materials are isotropic and orthotropic_layer. Each layer must specify a material, a thickness, and a fiber orientation.

```
Block 2
   TriaShell
   rotate 40 about axis 1
   rotate 15 about normal
   layer 1
     material 1
     thickness 0.01
     fiber orientation 40
   layer 2
     material 1
     thickness 0.04
     fiber orientation 44
End
```

Note that stress output can not be written to an exodus file for shells with more than one layer. However, layered stress values can be obtained by specifying stress in the **Echo** section.

3.10 Tria3

The **Tria3** is a three dimensional triangular shell with membrane and bending stiffness. There are 6 degrees of freedom per node. In most respects it is very

similar to the **TriaShell**. It is the default element for triangular meshes. The **Tria3** was provided by Carlos Felippa of UC Boulder. It currently supports only isotropic materials. It has a single required attribute, `thickness`, which may be specified in either the exodus file or the text input file.

The element stiffness matrix for triangles consists of the sum of two independent contributions from membrane and bending. These contributions may be arbitrarily scaled using the parameters **membrane_factor** and **bending_factor**. Each of these parameters default to 1.0. They must be specified in the text input file in the block definition.

Attribute	Keyword	Description
1	thickness	Thickness of the shell
2	offset	offset for the shell
N/A	membrane_factor	scale factor for membrane
N/A	bending_factor	scale factor for bending

The thickness may either be entered in the **Exodus** file, or in the input file. If an attribute is entered in both locations, the value in the input file will be honored. An example element block is shown below.

```
Block 3
        Tria3
        Thickness 0.01
        material 71
        membrane_factor=0  // turns off membrane stiffness
End
```

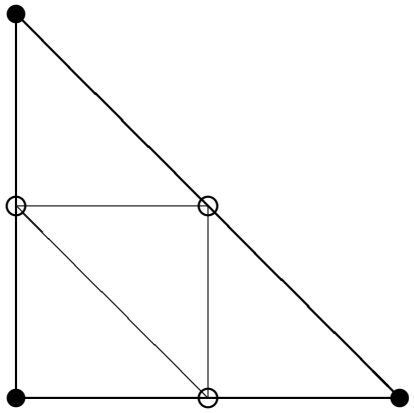
3.11 Tria6

The **Tria6** is a 6-node triangular shell with membrane and bending stiffness. The element properties and element stiffness and mass matrices are developed by internally generated **Tria3** elements (see Figure 10. It is not an optimal element, but is adequate for most applications. Shape functions are NOT quadratic. It is compatible with the **Quad8T** element, as well as with other elements based on the **Tria3**. See the description of the **Tria3** for details on the element.

3.12 Offset Shells

Any shell may be offset by specifying an offset. This single number is multiplied by the element normal to arrive at an offset vector. The resulting mass and stiffness

Figure 10: Tria6 Element
The element is generated by internally combining four Tria3 elements.



properties are equivalent to the stiffness generated by translating the shell by the offset vector, and constraining the resulting offset nodes to the untranslated nodes using rigid links. The performance is vastly better than the constraint approach. Note that for curved surfaces there may be modeling issues with offset elements since there is no change in curvature with the change in radius. In the `.inp` file the element offset is specified as,

```
offset=-3.14e-2
```

Offsets may also be specified in the exodus file. For shell elements these are specified in the attributes 2. Note however, that at this time there are few tools to support model building.

3.13 HexShell

The 8 noded hexshell is a hybrid solid/shell element. It is meshed as a standard hex element, but the formulation of the element is similar to that of a shell. Unlike a shell element, the thickness is determined by the mesh. But, the element is designed to operate with many of the same features as shell elements even when it becomes very thin. Details of the element formulation are available in a separate report (Ref. 18).

The hexshell has a preferential thickness direction which is essential to it's correct operation. The thickness direction may be specified in any one of three ways.[†]

[†] The element orientation may be identified in the output using the **eorient** keyword. See

1. Using the **tcoord**, it may be specified by a coordinate frame.

2. An exodus side set may be attached to one face of all the elements in a block using the keyword **sideset**. The thickness direction will be defined to be the normal to the sideset's surface. For example, if the sideset it placed on a side of the structure that lies on the x-y plane, then the thickness direction of the hexshell will be defined as the z direction, since that is the normal to the x-y plane.

3. Salinas may attempt to determine the thickness direction from the topology. This is the default option (because it is the easiest for the user), but it is also the least robust.

When the element thickness must be determined by the topology, the mesh must follow these requirements. The elements in the block must form a sheet. More than one disconnected portion of the sheet is possible, but all portions must adhere to these requirements.

- Every element in the sheet must have at least two neighbors, e.g. the sheet can't be a single element. NOTE... at this time, this is true for the parallel decomposed mesh as well. The portions of the sheets found in each subdomain can not be a single element. We must be able to eliminate the thickness direction of each element by it's neighbor connectivity.

- The elements in the sheet may vary in thickness, but the sheet must be exactly one element thick.

- The elements must be connected as a single sheet. Thus, if the sheet turns a corner, it must do so gently. The algorithm will fail if any element in the sheet is connected on the top or bottom to another element in the sheet.

The **HexShell** requires a material specification. Optional parameters include the sideset or the coordinate frame and coordinate direction used to determine the thickness direction. The sideset keyword must be associated with a defined sideset in the model. The **tcoord** keyword requires two integer arguments. The first is the ID of the coordinate system referenced. The second is the direction (1,2 or 3) associated with the coordinate system.

#	Keyword	Arguments	Description
1	sideset	ID	sideset to specify thickness direction
2	tcoord	ID and direction	coordinate frame and coordinate direction

section 2.8.23.

An example specification follows.

```
Block 88
   HexShell
   sideset 88
   layer 1
     material 1
     coordinate 1
     thickness .4
   layer 2
     material 2
     coordinate 2
     thickness 0.6
End

BLOCK 89
   HEXSHELL
   tcoord 5 1  // use coordinate frame 5, "x" direction
   material 89
END

BLOCK 100
  HexShell
  sideset 1 // the normal to sideset 1 will be the thickness direction for block 10
  material 1
END
```

The formulation of the HexShell supports multiple layers of orthotropic materials. Each layer has an associated material, normalized thickness and coordinate. The coordinate is provided to permit specification of the material coordinate. The thickness specifies the relative thickness of each layer. The total thickness is determined from the element topology, but relative thicknesses for each layer must be specified. If only one layer is specified, then the layer keyword is not required, and the relative thickness is irrelevant (and not required).

3.14 Beam2

This is the definition for a Beam element based on Cook's (Ref. 7) development. This beam is similar to the standard Nastran CBAR element. It has no shear contribution. The **Beam2** has 7 required parameters, and an optional offset vector.

#	old order #	Keyword	Description
1	1	Area	Area of beam
2	5	I1	First bending moment
3	6	I2	Second bending moment
4	7	J	Torsion moment
5,6,7	2,3,4	Orientation	orientation vector
8,9,10	8,9,10	offset	beam offset vector

No stress or strain output is available for beams. Beams are restricted to isotropic materials. Attributes may either be entered in the **Exodus** file, or in the input file. Attributes in the exodus file must be in the order specified in the table above. If an attribute is entered in both locations, the value in the input file will be honored. Two attribute orderings are currently supported in Salinas because of inconsistencies in preprocessing tools. See the discussion on "OldBeam" in section 2.3.

The following section illustrates the definition of a **Beam2** block.

```
Block 3
   Beam2
   Area 0.71
   I1 .05
   I2 5e-2
   J 0.994
   orientation 1.0 -1.0 0.9
   material 7
 End
```

Beams may be offset by specifying an offset vector. The resulting mass and stiffness properties are equivalent to a the stiffness generated by translating the beam by the offset direction, and constraining the resulting offset nodes back to the untranslated nodes using rigid links. Note that for curved surfaces there may be modeling issues with offset elements, since there is no change in curvature with the change in radius. In the `.inp` file the offset is specified as,

```
offset=-3.14e-2 0.11 0.99
```

Offsets may also be specified in the exodus file. For beams these are specified in the attributes `8, 9` and `10`. Note however, that at this time there are few tools to support model building.

3.15 OBeam

These beams are provided by Carlos Felippa of UC Boulder. They are similar to the simple beams of **Beam2**. They use identical parameters. Because of this duplication, these beams will probably be eliminated in the future.

3.16 Truss

This is the definition for a **Truss** element based on Cook (Ref. 7). Trusses have stiffness in extension only. The **Truss** has 1 parameter.

#	Keyword	Description
1	Area	Area of truss

No stress or strain output is available for trusses.

3.17 ConMass

Concentrated masses are used to apply a known amount of mass at a point location. Because many meshing tools build beams as a building block for **ConMass**, the geometry definition may be either a line or a point, i.e. the **Exodus** file element types are **BEAM**, **BAR**, **TRUSS** or **SPHERE**. If a line-type element is used, all the mass is associated with the *first* node of the element.

Parameters for the **ConMass** are listed below. Because of difficulties in translation or generation of the model, the parameters found in the exodus file are not normally used for a **ConMass**. This avoids the confusion generated when mass constant defaults may have been taken from beams for example. As a result, all parameters must be specified in the input or the analysis will fail.

This behavior can be tedious however, if many concentrated masses are found in the model, and if the analyst is confident that the attributes are appropriate for these elements. In this case, use the **ConMassA** element. It is identical to the ConMass, but uses the default attributes from the exodus file. Typically seven attributes would be specified there.

#	keyword	Description
1	Mass	concentrated mass
2	Ixx	xx moment of inertia
3	Iyy	yy moment of inertia
4	Izz	zz moment of inertia
5	Ixy	xy moment of inertia
6	Ixz	xz moment of inertia
7	Iyz	yz moment of inertia
8,9,10	offset	offset from node to CG

As an example element block,

```
Block 5
        ConMass
        Mass 1000.0
        Ixx 1.0
        Iyy 2.0
        IZZ 1.5
        offset 30.0 40.0 50.0
End
```

Note: *While offsets are provided for concentrated masses, their applicability depends on the model. In particular, an offset is meaningless if applied on a node for which there is no rotational degree of freedom. Conceptually, we are attaching the mass on the end of a long stiff beam. If that beam is attached only to a solid, it is free to rotate which is a model error. Salinas eliminates the offset in this case so the model is usable.*

3.18 Spring

The **Spring** element provides a simple spring connection between two nodes in a model. Note that the direction of application of the spring should be parallel to a vector connecting the nodes of the spring. It is usually preferable to have the nodes of the spring be coincident. Springs are defined in the exodus database using `BEAM` or `BAR` elements.

The **Spring** element has three required parameters (the translational spring stiffnesses). Rotational parameters are supported using the **RSpring** element described in section 3.19. Currently there is no way to attach off-diagonal elements, i.e. there is no K_{xy} spring element. If that is required, a combination of a spring and a multipoint constraint must be used.

Springs can be defined in user defined coordinate systems.

#	Keyword	Description
1	Kx	translational spring constant in X
2	Ky	translational spring constant in Y
3	Kz	translational spring constant in Z

As an example element block,

```
Block 51
      Spring
      Coordinate 7
      Kx 1e6
      Ky 1.11E7
      Kz 1000
End
```

3.18.1 Spring Parameter Values

It is strongly recommended that all three values of the spring constants be nonzero. This is especially important in parallel analysis performed using domain decomposition. Many domain decomposition tools may partition the model such that zero spring constants lead to singular domain stiffness matrices. This is true even if other elements may eliminate the singularity. This can cause the solver (particularly FETI) to fail.

While setting nonzero spring stiffness helps to avoid solver problems, the underlying domain decomposition problems still exist for parallel calculations. At the time of this writing, all available domain decomposition tools have difficulty with linear elements and particularly with springs. This invariably leads to load balance problems, and may introduce other problems. In many cases in large models, it may be better to replace the spring elements by solid element meshes which more accurately represent the physical connection. While there are more degrees of freedom in the calculation, the accuracy is enhanced, and domain decomposition problems are largely eliminated.

3.19 RSpring

The **RSpring** element provides a simple rotational spring connection between two nodes in a model. It is usually preferable to have the nodes of the spring be coincident. RSprings are defined in the exodus database using `BEAM` or `BAR` elements.

The **RSpring** element has three required parameters (the rotational spring stiffnesses). It is strongly recommended that all three components have some stiffness. This is particularly important when doing parallel analysis (see the discussion in section 3.18.1). Translational stiffness require the use of the **Spring** element described in section 3.18. Currently there is no way to attach off diagonal elements, i.e. there is no K_{xy} spring element. If that is required, a combination of an **RSpring** and a multipoint constraint must be used.

RSprings can be defined in user defined coordinate systems. The relevant parameters are listed in the table.

#	Keyword	Description
1	Krx	rotational spring constant in X
2	Kry	rotational spring constant in Y
3	Krz	rotational spring constant in Z

As an example element block,

```
Block 52
      RSpring
      Coordinate 7
      Krx=1e6
      Kry = 1.11E7
      Krz 0.1
End
```

3.20 Spring3 - nonlinear cubic spring

The **Spring3** element provides a nonlinear spring connection between nodes in a model. Note that the direction of application of the spring should be parallel to a vector connecting the nodes of the spring. It is usually preferable to have the nodes of the spring be coincident. Springs are defined in the exodus database using `BEAM` or `BAR` elements.

The **Spring3** element has nine required parameters (the translational spring stiffnesses). There is no way to attach off diagonal elements, i.e. there are no K_{xy} spring elements. If that is required, a combination of a spring and a multipoint constraint must be used.

The force applied by the **Spring3** is defined as a cubic polynomial in each of the coordinate directions. Thus,

$$F_x = Kx1 \cdot u_x + Kx2 \cdot u_x^2 + Kx3 \cdot u_x^3 \tag{30}$$

For linear analyses, only the first term is used.

Cubic springs may be defined in user defined coordinate system.

#	Keyword	Description
1	Kx1	translational linear spring constant in X
2	Ky1	translational linear spring constant in Y
3	Kz1	translational linear spring constant in Z
4	Kx2	translational quadratic spring constant in X
5	Ky2	translational quadratic spring constant in Y
6	Kz2	translational quadratic spring constant in Z
7	Kx3	translational cubic spring constant in X
8	Ky3	translational cubic spring constant in Y
9	Kz3	translational cubic spring constant in Z

As an example element block,

```
Block 51
      Spring3
      Coordinate 7
      Kx1 1e6
      Ky1 1.11E7
      Kz1 0
      Kx2 0
      Ky2 0
      Kz2 0
      Kx3 1e4
      Ky3 1.11E5
      Kz3 0
End
```

3.21 Dashpot

A dashpot represents a damping term proportional to velocity. Dashpot elements combine a viscous friction damper with a simple linear spring. The spring is included to avoid singular stiffness matrices when dashpots are connected without springs. Dashpots are currently only used in transient dynamic, direct frf and complex eigen analyses. For other analyses only the spring term will be used.

The damping factor is the damping matrix entry. It has units of *force·time/length*. For a single degree of freedom system with a mass=M, the following equation is satisfied.

$$K \cdot u + c \cdot \dot{u} + M \cdot \ddot{u} = f(t) \tag{31}$$

Currently dashpots are defined in the basic coordinate system only. Because they are single degree of freedom elements, the direction must also be defined (i.e. cid=1, 2 or 3). There are three parameters. All are required.

#	Keyword	Description
1	K	translational linear spring constant
2	c	damping factor
3	cid	coordinate direction (1, 2 or 3)

As an example element block,

```
Block 51
      Dashpot
      cid=1  // dashpot is in the X direction
      K=1e6
      c=1e5
End
```

Dashpots may be represented in the exodus file with any linear element. The Truss element most closely mimics the dashpot's single degree of freedom behavior, and may be the best definition for domain decomposition tools.

Caution should be exercised when using dashpots (or any single degree of freedom element). The remaining degrees of freedom must be properly accounted for, or the system matrices will be singular. Care should also be exercised to insure that if the nodes of the dashpot are not coincident, that the constraint force lies along the axis of the element - failure to do this can result in models that have nonzero rotational modes. There may also be important domain decomposition issues with dashpots. See section 3.18 for a discussion.

3.22 Hys

The **Hys** element provides a simple, one dimensional approximation of a joint going through microslip. Many simple joints can be represented by their *hysteresis* loop, a curve in the displacement vs. force plane. The relevant parameters of this element are indicated in the table, and illustrated in Figure 11.

#	Keyword	Description
1	Kmax	maximum slope of f vs u curve
2	Kmin	minimum slope of f vs u curve
3	fmax	maximum possible force
4	dmax	maximum possible displacement

The **fmax**, **dmax** pair define the limits of applicability of the element. The element will fail if the internal force exceeds **fmax** or the displacement exceeds **dmax**. The slope of the curve at the origin is **kmax**. It represents the small amplitude response of the system. The slope at the extremum, i.e. at (**dmax,kmax**) is **kmin**.

A **Hys** element uses a Beam or truss element in the exodus file. At the current time, the element may only be defined in the X direction. An example of the salinas input is shown below.

```
BLOCK 2
     Hys
     Kmax 4.5e+7
     Kmin 3.0e6
     fmax 5.92
     dmax 0.9833e-6
END
```

3.23 Shys

A **Shys** is the whole joint model developed by Smallwood and is an element which uses a Beam or truss element in the exodus file. The element is a 2.5 dimensional element with an **Shys** element in both the X and Y directions and a linear spring element in the Z direction. The **Shys** element is assumed identical in both the X and Y directions in this formulation. A coordinate system can be defined to orient the element correctly.

This element is being phased out in favor of the Joint2g element, where similar constitutive behavior can be specified if desired.

An example of the Salinas input is shown below.

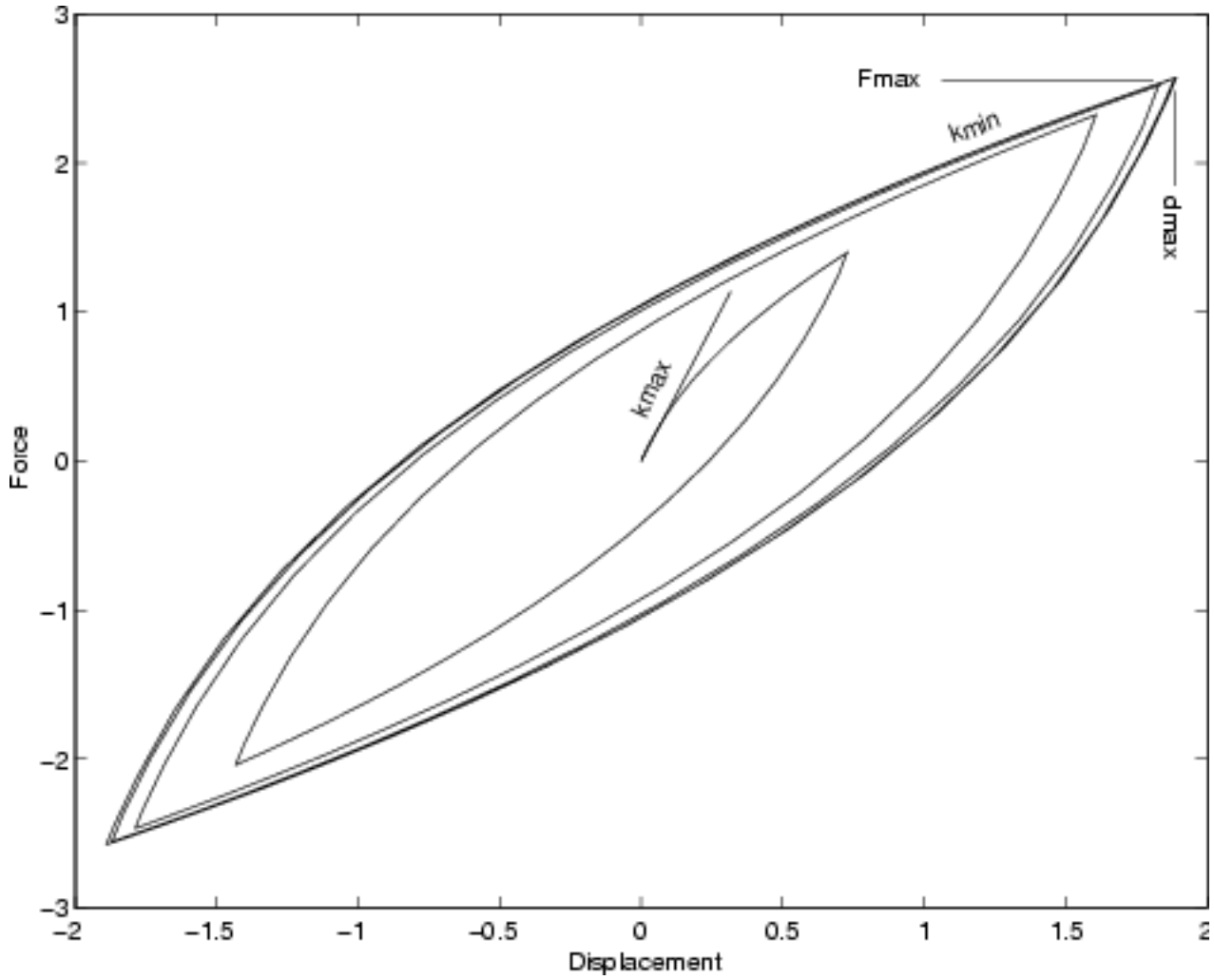


Figure 11: **Hys** element parameters

#	Keyword	Description
1	n	Exponent describing slope of force-dissipation curve at very small amplitudes
2	k	Linear stiffness of Smallwood's element
3	kNL	Coefficient for non-linear stiffness
4	kz	Linear translational stiffness in the Z direction
5	k_r	Linear rotational stiffness (optional, default = 0)

The **Shys** element does not use the attributes defined in the exodus file for default values of the optional parameters. A detailed discussion of the theory of the **Shys** element as well as how to determine the parameters can be found in the reports by Smallwood (Ref. 19).

```
BLOCK 2
     shys
     coordinate 2
     n = 1.39
     k = 1.3167e6
     kNL = 1.8499e6
     k_z = 1.6e6
     k_rot = 1.e9
END
```

3.24 Iwan

The Iwan model as a stand alone element has been phased out. Instead use the Joint2G element with an Iwan constituative model.

3.25 Joint2G

The **Joint2G** element† was devised to facilitate the implementation of "whole joint" models in Salinas. Beyond that it offers a workbench of considerable flexibility for specifying the nature of adherence between surfaces.

Each **Joint2G** element connects a pair of nodes (or *grids*, hence the "G" in *Joint2G*); it is a member of the geometrically one-dimensional class of elements *OneDim*. It's unique advantage is that it permits users to specify independently the constitutive behavior of each of the degrees of freedom connecting its node pair.

The constitutive behavior is implemented through a constitutive class that provides generalized scalar forces in response to corresponding generalized displace-

†Joint2G elements are supported and documented by Dan Segalman.

ments. Though the class name is *Axial*, members of the class provide responses that do not make reference to the axial or rotational nature of the deformation.

The decoupling of the constitutive response from the element machinery facilitates creating additional constitutive classes without having to recreate the whole element machinery.

The **Macroblock** provides a complementary functionality which may be used to specify the mechanically parallel behavior through the use of multiple, co-located **Joint2G** elements. See section 2.20.

3.25.1 Specification

The meshed objects that map into the **Joint2G** element are defined in the exodus database using BEAM or BAR elements. The Joint2G element does not make use of attributes defined in the exodus file; all properties must be specified in the BLOCK and PROPERTY cards. In the example below, properties are assigned to element block "2".

```
BLOCK 2
  coordinate 5
  joint2g
  kx=iwan    1
  ky=elastic 1.0e6
  kz=elastic 1.0e6
  krx=null
  kry=null
  krz=null
END
```

The above statement declares "BLOCK 2" to be of type **Joint2G**. It also declares the constitutive response in the "x" direction to be that of Segalman's 4-parameter Iwan model (SAND2002-3828). The parameters to be used in this model are those specified in "Property 1" defined below. In this case, the four parameters chosen are chi, phi_max, R, and S (χ, $\phi_{\max}$, R, and S in the SANDIA report). The Iwan properties can be specified alternatively by the parameter set chi, phi_max, F_S, and beta (χ, $\phi_{\max}$, F_S, and β).

```
property 1
    chi     =  -0.82139
    phi_max =   1.0325e-04
    R       =   7.608594e+06
    S       =   5.616950e+06
```

```
END
```

The constitutive behavior in the "y" and "z" directions is elastic with stiffness specified by the third argument - 1.0×10^6 in this case.

In this example, there is no specification for constitutive behavior in the three rotational directions. The *NULL* specification merely means that those degrees of freedom in the relevant nodes are are not activated ("touched") by this element. Because of artifacts associated with parallelization, it is recommended that if any of the rotational degrees of freedom are active (not *NULL*), they all should be active.

The directions ("x", "y", and "z") employed above are those associated with the coordinate system declared for the block. In the example shown, there is an explicit reference to coordinate system 5. If there is no such explicit reference to a coordinate system, then the "x", "y", and "z" directions are those of the global coordinate system.

3.25.2 Constitutive Behavior

4-Parameter Iwan Model (iwan): The Iwan element is a collection of spring slider elements designed to provide a predicted model of joint behavior (including energy loss). A detailed discussion of the theory of the Iwan element as well as how to determine the parameters can be found in the reports by Segalman (Ref. 20). Information about the Iwan element, and its relationship to other joint elements may be found in the Sandia internal report by Segalman and Starr (see 21).

The schematic of the Iwan model is shown in figure 12. Parameters for the behavior may be specified using either an older definition (Table 33), or a newer set (Table 34). The newer parameters are described briefly below, but the analyst is referred to the documentation for more detail.

#	Keyword	Description
1	chi	Exponent, χ, describing slope of force-dissipation curve at very small amplitudes
2	R	Constant coefficient in distribution
3	phi_max	Maximum break free pseudo-force
4	S	Strength of singularity in break free force distribution
	alpha	Geometric factor specifying nonuniform spacing of dphi (optional, default = 1.2)
	sliders	Number of slider elements (optional, default = 50)

Table 33: Older Iwan 4-parameter model

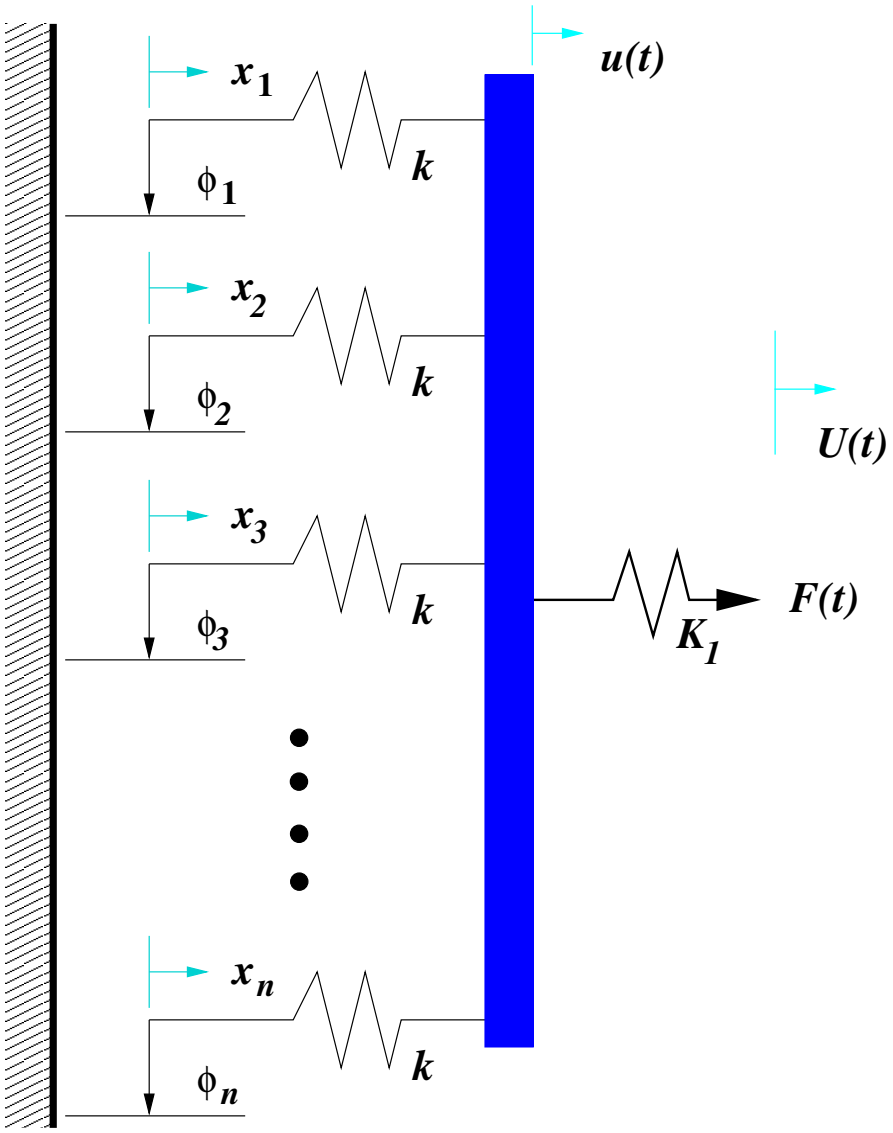


Figure 12: **Iwan** Constitutive Model

#	Keyword	Description
1	chi	Exponent, χ, describing slope of force-dissipation curve at very small amplitudes
2	beta	shape parameter of force/dissipation curve
3	KT	Tangent stiffness a very low loads
4	FS	Maximum break free pseudo-force
	alpha	Geometric factor specifying nonuniform spacing of dphi (optional, default = 1.2)
	sliders	Number of slider elements (optional, default = 50)

Table 34: Revised Iwan 4-parameter model

chi: determines the slope of the dissipation-force curve. Typically $0 < \chi < -1$. A value of zero corresponds to a coulomb type loss in Mindlin solutions. A value of $\chi = -1$ corresponds to a viscous like (but amplitude dependent) loss with dissipation proportional to the square of the amplitude. Dissipation follows the relation,

$$\text{Dissipation} \approx (\text{Amplitude})^{\chi+3}$$

beta: determines the shape of the dissipation-force curve. Larger β (say 5), produces power law behavior over all amplitudes. Beta affects both the shape of the hysteresis curve within microslip (Figure 13), and the abruptness of the transition from microslip to macroslip as shown in Figure 14. $0 \leq \beta < \infty$.

KT: determines the slope of the force-displacement curve at low amplitudes. This is equivalent to a spring constant, and is used as such in analyses for which the element is treated linearly.

FS: determines the force at which the last slider gives out, and element goes entirely into macroslip. The Iwan element is a statistical distribution of spring/slider elements. This is a point on that distribution.

Smallwood's Hysteresis Model (shys): D.O. Smallwood developed a three parameter model that captures the power-law behavior of energy loss with force amplitude. The model parameterizes the hysteresis loop determined from experimental data in such a way that the power law behavior is preserved.

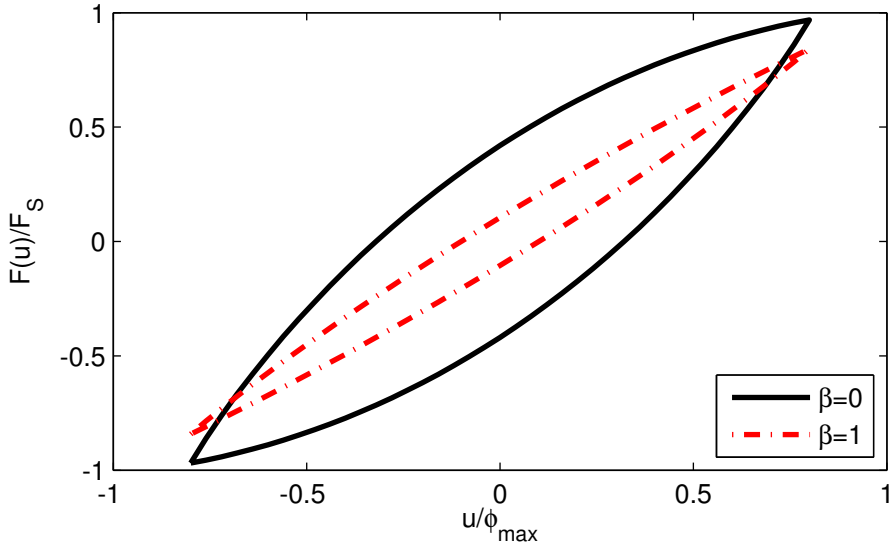


Figure 13: Dimensionless hysteresis curves for the four-parameter Iwan model with $\chi = -1/2$ and two values of β.

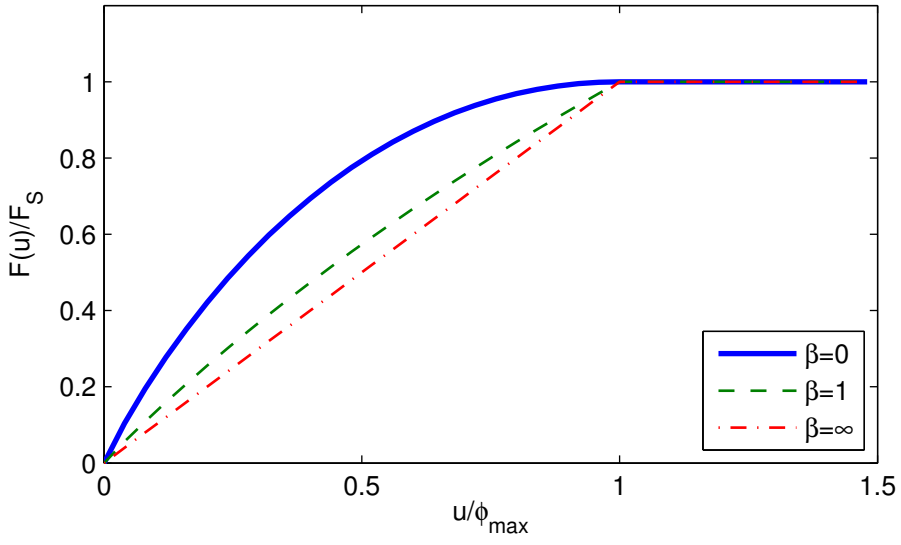


Figure 14: Dimensionless static loading curves for the four-parameter Iwan model with $\chi = -1/2$ and three values of β, as the model goes into macroslip.

#	Keyword	Description
1	n	Exponent describing the slope of the force dissipation curve at small amplitudes
2	k	Coefficient for the linear stiffness
3	knl	Coefficient for the non-linear stiffness

A detailed discussion of the theory of the **shys** model as well as how to determine the parameters can be found in reference 22.

```
PROPERTY 1
    n   =  1.39
    k   =  1.3167e6
    knl =  1.8499e6
END
```

One Dimensional Gap Model (gap): The **gap** model attempts to represent the behavior of a gap closure with a bilinear elastic element. For proper numerical behavior, the stiffness of the open gap should not be more than a few orders of magnitude less than the stiffness when the gap is closed. The **Joint2G** implementation of the **gap** model is identical to the axial behavior of NASTRANS cgap/pgap element as well as the axial behavior of the stand alone version of the gap element implemented in Salinas (section 3.26).

#	Keyword	Description
1	Ku	Unloaded Stiffness
2	Kl	Loaded Stiffness
3	U0	Initial Gap Opening
4	F0	Preload (force at U0)

```
PROPERTY 1
    ku  =  1e5
    kl  =  1e6
    U0  =  0.01
    F0  =  200
END
```

Elastic Plastic Hardening Model (eplas): The **eplas** element is an elastic-plastic 1-dimensional element with linear isotropic hardening. Both the plastic strain

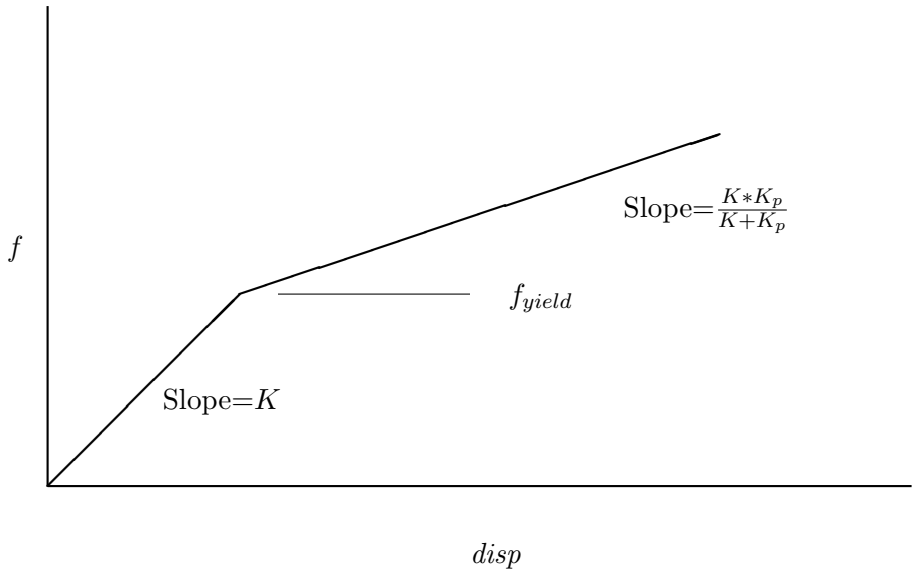


Figure 15: Eplas Model

and the hardening variable are initialized to zero. The parameters are illustrated in Figure 3.25.2.

#	Keyword	Description
1	k	Linear Stiffness
2	kp	Hardening Stiffness
3	fyield	Force at Yield

```
PROPERTY 1
    k       =  1e6
    kp      =  1e5
    fyield  =  1e4
END
```

One Dimensional Spring-Dashpot Model (damper): A **damper** represents a damping term proportional to velocity. Damper elements combine a viscous friction damper with a simple linear spring. The spring is included to avoid singular stiffness matrices when dampers are connected without springs. Dampers are currently only used in transient dynamic, direct frf and complex eigen analyses. For

other analyses only the spring term will be used. The behavior of this element is identical to **dashpot**.

The damping factor is the damping matrix entry. It has units of *force·time/length*. For a single degree of freedom system with a mass=M, the following equation is satisfied.

$$K \cdot u + \mu \cdot \dot{u} + M \cdot \ddot{u} = f(t) \tag{32}$$

#	Keyword	Description
1	K	Stiffness
2	Mu	Viscous Damper Coefficient

```
PROPERTY 1
    K   =  1e6
    Mu  =  1e2
END
```

Additional Constitutive Behavior: The philosophy employed in the implementation of the **Joint2G** element of decoupling the constitutive behavior from the element machinery should facilitate the implementation of other constitutive models. Among those whose implementation is foreseen are the following:

- Bouc-Wen hysteresis model
- Preisach hysteresis model

3.26 Gap

Gap elements are modeled after the non-adaptive nastran `CGAP/PGAP` elements. They are intended to provide a simple, penalty type element suitable for modeling simple connections. Note that these elements (like all beam-like elements) when embedded in solid meshes can result in difficult domain decompositions, and lead to load imbalance.

The **Gap** element is inherently nonlinear. In linear analysis, the element behaves approximately like a spring with the stiffness determined by `KL` and a transverse stiffness, KT. The parameters of the element are listed in the table below and shown graphically in Figure 16.

#	Keyword	Description
1	KU	unloaded stiffness
2	KL	loaded stiffness
3	KT	transverse stiffness
4	U0	initial gap opening
5	F0	Preload, i.e. force at U0
6	coordinate	Required coordinate frame.

The unloaded stiffness, KU, represents the stiffness of the element when the gap is open. It must be greater than zero. The loaded stiffness, KL, represents the stiffness when the gap is closed (as shown in the figure). The stiffness is KL when UA - UB is greater than U0.

The initial gap opening and preload define the corner point in the force/deflection curve as shown in Figure 16. Typically these will be zero.

A gap element provides for transverse stiffness and friction. When the gap is closed, the transverse stiffness is KT. If the gap is open, the transverse stiffness is reduced to $KU' = KU \times KU/KL$.

The coordinate frame is a required attribute of the gap element. The gap open and closes along the X axis of the frame. Note that the direction of the coordinate frame is quite important. The element determines a quantity $UA - UB$ along this coordinate axis. *This axis may not align with the coordinate alignment of the elements, which can lead to confusion.*

The gap element is a simple penalty type element that somewhat mimics the effect of a physical gap. Choice of the value of KL is very important to success of the element. Good values are somewhat in the range of the neighboring element stiffness. Too large a value can lead to matrix condition problems. Too small a value results in excessive softness and penetration in the gap.

Because the element is nonlinear, it has a significant impact on solutions. As described in section 2.1.17 (and the **update_tangent** keyword), the default behavior for the nonlinear solver is a partial Newton iteration. This means that the tangent stiffness matrix is not updated between iterations. Thus, if KL and KU are quite different, the solver will be using the wrong slope in the newton loop. Many, many iterations may be required for convergence. You may want to turn on the 'nl-residual' option in the echo section (see 2.7) which will put convergence information into the results file.

An example is shown below.

```
BLOCK 2
     GAP
     KL 4.5e+7
```

```
      KU 3.0e6
      KT=1e6
      f0 5.92
      u0=0.9833e-6
      coordinate 5
END
```

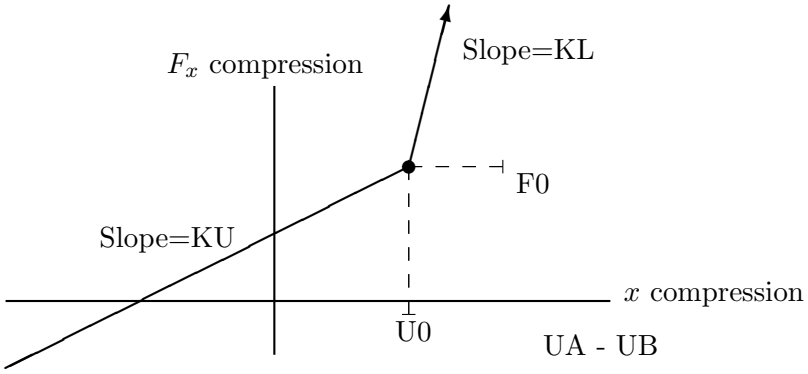


Figure 16: Gap element Force-Deflection Curve

Gap Issues. The gap element is definitely more complex than most elastic elements. Here is a partial list of "gotchas" that we have observed.

- Gaps should normally be zero length elements. Like springs, a gap that has a physical length will not be invariant to rigid body rotation. See section 5.2.3. One approach to this would be to use a combination of beam and gap elements.
- The gap element uses a coordinate frame to define its direction. *The direction is NOT set by the nodal coordinates.*
- The direction of the gap element must correlate to the displacement difference from $UA - UB$. It is very easy to get this direction reversed.
- If you set **U0**, you must also set **F0**. This element does not constrain the force/displacement curve to go through zero. The input must do this. The gap element may thus be used to enforce an initial displacement or force. That may not be what you want. It can cause very slow convergence on the initial time step.

- Significant numerical damping may be required for convergence. Closing the gap can cause energy to be moved into higher frequencies. Without numerical damping, this energy can multiply until the solution becomes unstable. Numerical damping is best introduced by setting "`rho`" in the time integrator. Values of "`rho=0.2`" to "`rho=0.7`" have worked well. It is problem dependent.

  Physically closing a gap would cause some energy loss, either by microslip, or by a small amount of local plastic deformation. Numerical damping can dissipate this energy that is removed from the physical system by means that are not included in the finite element model.

- This gap element may not conserve energy. This is demonstrated in Figure 17, where a mass is dropped onto a gap. A completely elastic rebound would take the mass back to zero. Instead, it rebounds significantly above zero. This issue comes about because of time discretization. The mass "penetrates" the gap region too far, which stores too much energy in the element. It is then expelled with too much velocity. The only solution with this element is to reduce the integration step.

3.27 Gap2D

The **Gap** element of the previous section provides a useful construct for planar type interactions. A common modeling issue is a bolt in an oversized hole. To model this interaction an ellipsoidal gap element (or **Gap2D**) may be required.

The **Gap2D** element operates just like the Gap element except that the gap could open in 2 dimensions. The gap is open provided that the element displacement is within an ellipse defined by the major and minor axes.

$$\left(\frac{u_x}{\text{U0X}}\right)^2 + \left(\frac{u_y}{\text{U0Y}}\right)^2 < 1 \tag{33}$$

The major and minor axes of the ellipse are defined in the x and y direction of the *required* coordinate frame.

Parameters of the **Gap2D** element are listed below.

#	Keyword	Description
1	KU	unloaded stiffness
2	KL	loaded stiffness
3	KT	transverse stiffness (z direction)
4	U0X	initial gap opening, major direction
5	U0Y	initial gap opening, minor direction
6	coordinate	Required coordinate frame.

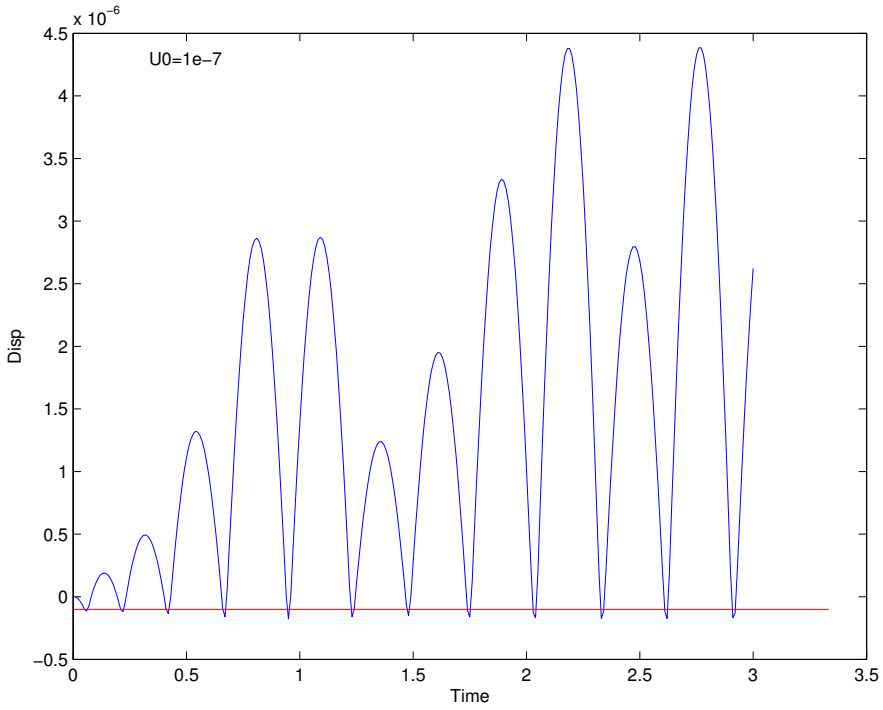


Figure 17: Mass bouncing off a Gap

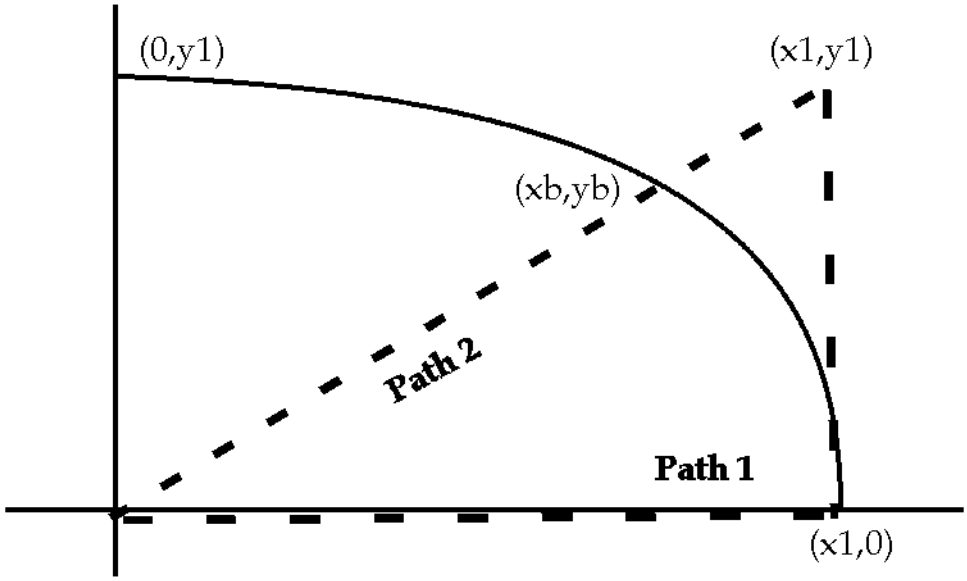


Figure 18: Gap2D force diagram

While the gap geometry is defined as an ellipse, the stiffnesses are not. In the open section of the element, the in-plane stiffness is `KU`, and is independent of direction. Likewise, in the closed gap region, the in-plane stiffness is independent of direction, and is defined by `KL`. The out of plane stiffness for this element is *always* `KT`. Note that the transverse stiffness behavior is significantly different than that of the standard **Gap** element.

The definitions above define the gradient of the force only, and for this nonlinear force, the value of the force depends on the path chosen for integration. For this element, we define the force as the integral along the shortest line from the origin.

In Figure 18, two possible integration paths are shown for arriving at the point (x_1, y_1). In the first path, we integrate to $(x_1, 0)$ and then up to (x_1, y_1). The y component of force is $f_y^{(1)} = \texttt{KL} \cdot y_1$. In path 2, we follow the straight line through (x_b, y_b). The associated force is $f_y^{(2)} = \texttt{KU} \cdot y_b + \texttt{KL}(y_1 - y_b)$. For this element, we always choose the shortest line path (path 2). This insures that the force is not history dependent.

3.28 GasDmp

The **GasDmp** element is a nonlinear, beam-like element that simulates the damping forces on MEMS devices due to gas pressure as MEMS beams vibrate. The element has no stiffness, but has damping roughly proportional to velocity/L^3, where L is the distance from the beam to the substrate. The element is very experimental, and still under development. Contact Troy Skousen or Burak Ozdoganlar at 845-0427 for details.

Inputs to the **GasDmp** element are as follows.

#	Keyword	Description
1	W	Beam width (length units)
2	dL	Considered length of beam (length)
3	mm	Molecular mass of gas (mass)
4	p0	Ambient pressure of gas (pressure)
5	T	Ambient temperature of gas (temperature)
6	muRef	Reference viscosity (pressure * time)
7	TRef	Reference temperature (temperature)
8	ww	Viscous temperature exponent

Currently all of the parameters are implemented through the input file and not through the Exodus_II file. At a future date the beam width and length will be tied to the mesh.

The theory for the development can be found in an internal Sandia draft report available on the Sandia internal web at,
http://www.jal.sandia.gov/Salinas/external-reports/microbeam2.pdf
Most of the implementation is associated with equations 9 and 10 of this report.

3.29 MPC

Multi-Point Constraints (or **MPC**s) are constraint equations applied directly to the stiffness matrix. They are not elements, and are not available from an **Exodus** database. However, in many respects they look like elements, and can be thought of as elements. Some analysis codes treat them as pseudo elements.

All **MPC**s describe constraint equations of the form,

$$\sum_i C_i u_i = 0$$

where C_i is a real coefficient, and u_i represents the displacement of degree of freedom i.

Unlike many Finite Element programs, **Salinas** does not support user specification of constraint and residual degrees of freedom (DOF). In serial solvers the partition of constrained and retained degrees of freedom is performed simultaneously by gauss elimination with full pivoting so the constrained degrees of freedom are guaranteed to be independent. In parallel solvers (such as FETI), the constraints are specified as lagrange multipliers which involve no such partitionings. Redundant specification of constraint equations is handled by elimination of the redundant equations and issue of a warning. User selection of constrained DOF in Nastran has led to significant headaches for analysts who must insure that the constrained DOF are independent and never specified more than once.

Each **MPC** is specified in the input file with a section descriptor. Note that a separate section is required for each equation (or degree of freedom eliminated). An optional coordinate system may be specified on the input, but must be the first entry in the section[‡]. The **MPC** will be stored internally in the basic coordinate system (coordinate frame 0). The input consists of a triplet listing the global ID of the node, a degree of freedom string, and the coefficient of that degree of freedom. The degree of free strings are x, y, z, Rx, Ry, Rz. They are case insensitive.

#	Keyword	Description
1	coordinate	optional coordinate frame with *integer* id
2	*integer*	*integer* node number in global model (The node number MUST USE 1 TO N ORDERING like exodus file numbering).
3	dof string	*string* x, y, z, Rx, Ry, or Rz
4	coefficients	*Real* weight associated with this dof
items 2-4 may be repeated as many times as needed		

In the following example, the x and y degrees of freedom in coordinate system 1 are constrained to be equal for node 4.

```
MPC
    coordinate 1
    4 x 1.0
    4 y -1.0
END
```

[‡]At this time, all the nodes in an MPC must be associated with the *same* coordinate system.

IMPORTANT

Constraints are handled in various ways by the linear solvers. In the serial solver, the dependent degrees of freedom are eliminated before the matrices are passed to the solver. In parallel, we use lagrange multipliers to handle the constraints. There is currently no user control of constraint handling methods.
Note also that there are practical differences between rigid elements (described in the following sections) and constraint equations that are nominally identical. For parallel solutions, we are currently using an augmented lagrange type solution method with the rigid links. This means that terms are added to the stiffness matrix in parallel with the constraints. In most cases, this renders the matrices positive definite, and greatly increases robustness and solution performance with no penalty for accuracy. Thus, rigid links are recommended whenever possible in parallel solutions.
Finally note that replacing rigid links with very stiff beams can be a bad thing to do. The condition of the resulting matrices can be severely degraded which can lead to significant loss of accuracy.

3.30 RROD

An **RROD** is a *pseudo*element which is infinitely stiff in the extension direction. The constraints for an **RROD** may be conveniently stated that the dot product of the translation and the beam axial direction for a **RROD** is zero. There is one constraint equation per **RROD**.

The **RROD** is specified using beams or trusses in the **Exodus** database, with a corresponding **Block** section in the salinas text input file. No material is required and any number of connected or disconnected **RROD**s may be placed in a block. The following is an example of the input file specification for **RROD**s if the **Exodus** database contains beams in block id=99.

```
Block 99
    RROD
END
```

3.31 RBar

An **RBAR** is a *pseudo*element which is infinitely stiff in extension, bending and torsion. The constraints for an **RBAR** may be summarized as follows.

1. the rotations at either end of the **RBAR** are identical,
2. there is no extension of the bar, and
3. translations at one end of the bar are consistent with rotations.

The **RBAR** is specified using beams or trusses in the **Exodus** database, with a corresponding Block section in the input file. No material is required and any number of connected or disconnected **RBAR**s may be placed in a block. The following is an example of the input file specification for **RBAR**s if the **Exodus** database contains beams in block id=99.

```
Block 99
    RBAR
END
```

RBARs can be reordered so that the number of **RBAR**s connected to a single node is minimized. Having a large number connected to the same node results in a highly populated matrix and a slow computation. Therefore, reducing the number of connections can shorten runtime. (see the *reorder_rbar* parameter in the **PARAMETERS** section 2.3).

3.32 RBE2

Salinas has no support for the Nastran **RBE2** element. However, in most cases there is little difference between the **RBE2** element and a collection of **RBAR**s.

3.33 RBE3

The **RBE3** pseudo-element's behavior is taken from Nastran's element of the same name. Two distinct versions of the element are available, but the older version will be deprecated sometime in the future. Each method is each described below, with significantly more detail found in section 2.15.3 of the theory manual . The element is used to apply distributed forces to many nodes while not stiffening the structure as an **RBAR** would. The **RBE3** uses the concept of a slave node.

Because all the nodes in an **RBE3** are not equivalent, each **RBE3** requires its own block ID. In the **Exodus** file, all links connecting to a single **RBE3** are defined in a single element block. The input file then specifies that this is an **RBE3** element block, as shown in the example below. If the model requires many **RBE3**s, a separate block must be specified for each.

Usage. The optional parameters for the Rbe3 pseudo-element are shown in the table below. These parameters must be specified in the input file, not as attributes of the exodus file.

Keyword	**value**	Description
refc	*string*	reference coordinates for slave
method	*new* or *old*	Constraint computation method
WT	6 *reals*	relative weight of coordinates

refc. The **REFC** parameter sets the degrees of freedom to activate on the slave node. The keyword **REFC** provides a text representation of the active degrees of freedom involved in the constraints. Thus, `REFC='12'` provides 2 equations that constrain degrees of freedom associated with X and Y translations. No other degrees of freedom are affected. If the **REFC** keyword is not provided, it defaults to `REFC='123456'`, i.e. constraint relations will be provided for all 6 structural degrees of freedom on the slave node.

method. This parameter determines which formulation is used to determine the constraint relations. By default, the *new* method is used in versions of Salinas newer than 2.0. See below.

WT. The contributions of each of the coordinates of the independent nodes may be scaled by **WT**. Most typically this would be used to determine the relative weight of rotational degrees of freedom on the independent nodes to the computation of the slave node rotations. The default value is **WT**= 1 1 1 0 0 0 which means that the rotations do not contribute to the Rbe3.

Generally we recommend there be no contribution from the rotations. The rotation of the element may then be determined solely from the translational degrees of freedom on the independent nodes.

The parameter applies only to the new method. In the old method rotations on the independent nodes are always ignored.

New Method Rbe3. The new formulation of the Rbe3 is based directly on the published method from MSC nastran. Details of the method are described in section 2.15.3 of the *theory manual*.

Old Method Rbe3. Previous to version 2.0, a version of the RBE3 was generated based on an *ad hoc* mathematical approach. This element should act like a Nastran **RBE3** for most applications, but its use is discouraged.[§]

Cautions in using RBE3. While a very convenient construct, the **RBE3** is not a true element, and it can introduce complexity in the solution. Following are a few things to bear in mind in using the element.

- Very large RBE3 elements may spread across a large portion of the model. This affects linear solvers that are typically designed to propagate error locally. As a consequence convergence may be slow.
- Large RBE3 elements may require a lot of memory. This memory is stored on a single processor.
- No MPC should be linked to another. Many of our solvers will fail if one MPC type element shares nodes with another.
- Prescribed accelerations (see section 2.12.2) cannot be applied on an **RBE3** or any other MPC.
- The element has no logic to determine which degrees of freedom of the independent nodes are active. Thus, if you specify $WT = 1\,1\,1\,1\,1\,1$ the element will try to determine it's rotation based on a combination of the translational and rotational degrees of freedom on the independent nodes. If the rotational degrees of freedom are inactive, they are treated as zero. This is rarely what is wanted.
- Care must be taken to insure that only one node of the **RBE3** has multiple connections to its links. Further, all links in the **RBE3** must be connected to the slave node.
- We note that many of our trouble tickets come from Rbe3 elements.

[§] These elements are not identical. In particular, RBE3 elements that have 2 or fewer dependent nodes, or for which the dependent nodes are colinear will either not work, or not work as anticipated. As outlined in the theory manual, the rotational degrees of freedom on the independent nodes are ignored. Further, the old formulation will differ from the new approach if the slave node is far from the centroid of the element.

Example Rbe3. The following is an example of the input file specification for an **RBE3** if the **Exodus** database contains beams in block id=99.

```
Block 99
    RBE3
    refc=123456
    method=new
    wt=1 1 1 0 0 0
END
```

3.34 Superelement

Superelements have various meanings in commercial codes. **Salinas** does not support a full automatic superelement capability. In section 2.1.7 the procedure for reducing an entire model to a reduced order model is outlined. Import of a such a reduced model (or superelement) into **Salinas** is also supported. The **superelement** described in this model involves import of a mass and stiffness matrix into a full system model. This linearized approach complements a Craig-Bampton (and other) reductions, and may be used in any type of analysis.

Limitations

- The superelement must be small enough to fit on a single processor in a parallel run. No consideration for superelements which span processors is made.
- Nodes on the superelement interface may be shared across processors. Interior degrees of freedom are local to a single processor.
- Output of the interface node degrees of freedom will be made in the base model in the usual way. Output of internal superelement quantities will be made in the superelement database file.
- No automatic data recovery is available.
- Only a single level of superelement is supported.

User Input

Each superelement must be placed a unique block, i.e. there is one superelement per block. The following input is provided by the user.

connectivity: To provide the geometric connectivity to the model, the connectivity must be added to the exodus file. If the superelement has the same number of nodes as a standard element, the analyst may choose to use such an element to provide the connectivity. This can facilitate visualization of the model. When the model is larger, a tool is provided to directly add the superelement to the exodus database.¶

Note that codes such as nastran input superelements by connecting to the nodes directly. In a parallel environment, it is critical that the superelement remain on a processor. As a consequence, the decomposition tool must have knowledge of the superelement. It must therefore be in the finite element database. This is also consistent with the other tools used with **Salinas** where node numbers are not typically provided directly. This permits insertion of the superelement in a part, with a subsequent node reordering from `gjoin` for example.

Salinas does support an element with *more* nodes than required for the connectivity map. Thus, a Hex-8 could be used to define the connectivity for a superelement with 7 nodes on the interface. Obviously the connectivity map cannot have more nodes than the element.

connectivity map: The equations for the system matrices must be associated with the nodes and degrees of freedom in the model. The following example creates a map for an eight degree of freedom reduced order matrix. The first column of the map is associated with the node index in the element. The second degree of freedom defines the coordinate direction (typically 1 to 6 for x, y, etc).

```
// node  cid
 map 0    0
     0    0
     1    1
     1    2
     1    3
     2    1
     2    2
     2    3
```

In this example, the first two rows of the system matrices are associated with internal degrees of freedom. These interior dofs are indicated by a zero for both the node index, and the coordinate direction. Row 3 of the matrix is

¶The tool is named "**mksuper**". It is part of our standard tools distribution.

associated with the first node in the element connectivity, and with the x coordinate direction. Row 8 is associated with the second node, and the z coordinate direction.

There must be exactly as many rows in the connectivity map as there are rows in the system mass and stiffness matrices.

If the node index is less than zero, the row of the matrix associated with that degree of freedom will not be mapped to the system matrix. This can be used to "clamp" a generalized degree of freedom.

> *The node index is NOT the node number in the exodus file. Rather it is the index into the element connectivity. Thus, for a four node element, the index must never exceed 4. This permits the use of* `gjoin` *and other tools without the need to reorder these terms in the input file.*

system matrices: The system matrices must be provided in a **netcdf** file. Tools will be provided to create this file, and the format of the file will be documented with those tools. The file will contain the following.

Kr. The reduced stiffness matrix. This is required for all analysis.

Mr. Most analyses require a reduced mass matrix as well. It's dimension must match that of the stiffness matrix.

Cr. A reduced damping matrix may be used for some analyses. It is entirely optional, but if present, must be of the same dimension as **Kr**.

A good reduced **Kr** for 3D analysis should have exactly 6 zero energy modes. It must be symmetric (**Salinas** will try to symmetrize it). Typically **Mr** would be nonsingular. Failure to meet these requirements can confuse the entire solution procedure, and lead to erroneous solutions.

transfer matrices: Output of results on interior points in the superelement are facilitated using optional output transfer matrices (OTM). These are described in some detail in the section on model reduction (2.1.7). These matrices are used *only* if superelement output is requested in the output specification. The following matrices apply.

OTM Nodal output transfer matrix.

OTME Element output transfer matrix.

OutMap An optional node map for the OTM.

OutElemMap An optional element number map for OTME.

output specifications: Output from superelements may be requested in the "ECHO" or the "OUTPUT" sections. If requested in the "OUTPUT" section, then a new exodus file will be generated based on the information and name of the netcdf file. The number of nodes in the new file is the sum of the number of nodes on the interface and the number of nodes in the output transfer matrix, OTM. The number of elements is the number of elements in the OTME. All elements will be placed in a single element block. For either the echo or the output sections, output of superelement data is specified by the **superelement** keyword.

Because we don't know the connectivity of the elements in the OTME, all such elements will be defined as sphere elements, and will be collocated on a single node in the model. This makes visualization pretty much useless, but the element data is preserved for other types of post processing.

Likewise, no coordinate information is available for the interior nodes of the model. These elements will be located at the origin of the system.

Parameters

The parameters for the superelement block are listed in the table.

Keyword	**value**	Description
map	*ints*	table of node/cid pairs
file	*string*	netcdf file containing matrices
savememory	*yes* or *no*	controls storage of matrices in memory
diagnostic	int	0 = run no diagnostics 1 = compute Kr * RBM 2 = compute eig(Kr,Mr)

Block Example

The above parameters are entered in the **block** section of the input file. For example,

```
BLOCK 10
  superelement
  file='example.cdf'
  // node  cid
```

```
    map 0    0
        0    0
        1    1
        1    2
        1    3
        2    1
        2    2
        2    3
   diagnostic=1
END
```

3.35 Dead

A **dead** element has no mass and no stiffness. It may be of any dimensionality, solid, planar, line or point. Interior nodes to a block of **Dead** elements will not be included in the computation of the model. There are no parameters for **Dead** elements.

4 Stress/Strain Recovery

Stresses and strains are recovered at the centroids of the finite elements using standard finite element procedures. Stress and strain recovery is not implemented for 1-D elements. The stresses/strains calculated for shell elements are calculated in element space and not global space.

4.1 Stress/Strain Truth Table

The exodus data format provides an element truth table. Element variables are defined globally (for all element blocks), but output data is stored only for those blocks that have entries in the truth table. Thus, in Salinas if stress output is requested (see section 2.8.10), then stress variables are defined for solids and shells.[†] Space is allocated in the output exodus file, and data is written only if it is applicable. Table 35 illustrates this for stresses. A similar table can be generated for strains. Note that volume stresses always start with "V" and surface stresses start with "S". Note that "vonmises" is the only entry that applies to both solids and shells.

4.2 Solid Element Stress/Strain

If stresses are requested, solid elements will output the values of stress at the element centroid.[‡] The values reported are the engineering stresses in the globabl coordinate frame. That is,

$$\sigma_{ij} = \sum_{k,l} D_{ijkl}\epsilon_{kl} \tag{34}$$

Where D_{ijkl} is the material tangent modulus tensor, and

$$\epsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right).$$

Here u and x are the displacement and coordinates in the basic coordinate frame.

4.3 Shell Element Stress/Strain

Shell elements introduce two complexities to stress/strain recovery. First, it is often important to recover data from the virtual surfaces of the elements (where the stresses are highest). This requires data recovery at the top, midplane and bottom

[†] The variables are defined for solids and shells even if only one or the other occurs in the model

[‡] There is little point in reporting stresses elsewhere in the element as none of the post processing tools currently available properly manage stresses except at the centroids.

Table 35: Element Stress Truth Table

Variable Name	**Element**		
	Solid	**Shell**	**Beam**
SStressX1		σ_{xx}^{top}	
SStressY1		σ_{yy}^{top}	
SStressXY1		τ_{xy}^{top}	
SvonMises1		σ_{vm}^{top}	
SStressX2		σ_{xx}^{mid}	
SStressY2		σ_{yy}^{mid}	
SStressXY2		τ_{xy}^{mid}	
SvonMises2		σ_{vm}^{mid}	
SStressX3		σ_{xx}^{bottom}	
SStressY3		σ_{yy}^{bottom}	
SStressXY3		τ_{xy}^{bottom}	
SvonMises3		σ_{vm}^{bottom}	
VStressX	σ_{xx}		
VStressY	σ_{yy}		
VStressZ	σ_{zz}		
VStressYZ	σ_{yz}		
VStressXZ	σ_{xz}		
VStressXY	σ_{xy}		
VonMises	σ_{vm}	$\max(\sigma_{vm})$	
ElemForce			forces

surfaces. Second, there are no stresses or strains normal to the surface. Thus, stresses are naturally reported in the surface of the element. This can also introduce confusion about the inplane coordinate frames. As shown in Figure 19, the stresses and strains are recovered in the physical space x_1, x_2 coordinate frame, which has been mapped from the η_1, η_2 frame in element space. Note that the direction of the x_1 vector depends on the ordering of the mesh, and may vary from element to element in the same surface mesh. The element orientation vectors can be obtained with the **eorient** keyword described in section 2.8.23. The von mises stress, will of course be independent of the element orientation vectors (as it is an invariant).

The **TriaShell** stress recovery is described here. The **TriaShell** is a shell element created by combining Allman's triangle[16] and the DKT element.[17] The stress vector for the element is $\vec{\sigma} = (\sigma_x, \sigma_y, \sigma_{xy})^T$. This can be further broken down as:

$$\vec{\sigma} = \vec{\sigma}_{at} + \vec{\sigma}_{dkt} \tag{35}$$

where $\vec{\sigma}_{at}$ is the stress vector for Allmans's triangle and $\vec{\sigma}_{dkt}$ is the stress vector for the DKT element. Since Allman's triangle represents the membrane d.o.f., i.e., (u, v, θ_z), the stresses through the three surfaces of the shell element are the same. Therefore,

$$\vec{\sigma}_{at} = [D]\{\epsilon\} \tag{36}$$

where $\{\epsilon\}$ is the strain vector, and $[D]$ is the elasticity matrix for Allman's triangle. For the DKT element,

$$\vec{\sigma}_{dkt} = z[D]\{\kappa\} \tag{37}$$

where z is the coordinate direction normal to the element, with $z = 0$ representing the mid-plane, $[D]$ is the elasticity matrix for the DKT element, and

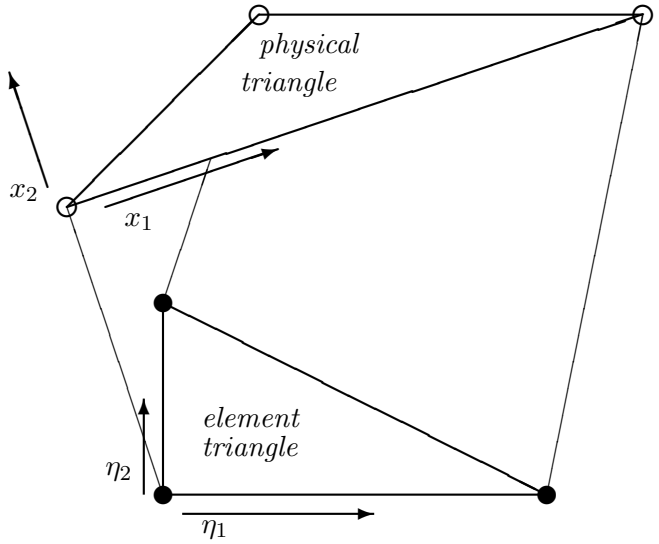
$$\{\kappa\} = \begin{bmatrix} \beta_{x,x} \\ \beta_{y,y} \\ \beta_{x,y} + \beta_{y,x} \end{bmatrix} \tag{38}$$

where β_x and β_y are rotations of the normal to the undeformed middle surface in the x-z and y-z planes, respectively (assuming the element lies in the x-y plane). $\vec{\sigma}_{dkt}$ does vary with the thickness of the element. Note, the above stress equations are written with respect to a local element coordinate system as shown in Figure 19.

Combining the stress vectors from Allman's triangle and the DKT element above yields the stress vector for the element which is output in the local element frame.

For composite elements (such as QuadT, Quad8T and Tria6), the stresses are computed from the underlying Tria3 element and then transformed to the element orientation of the composite element. For the quad elements, the stress of the two central triangles is averaged. Figures 8, 9 and 10 describe these composite elements.

Figure 19: Tria3 Stress Recovery. Stresses are output in the orthogonal x_1, x_2 coordinate frame in physical space, which has been mapped from the η_1, η_2 frame in element space.



4.4 Line Element Stress/Strain

Reporting stresses for line type elements (Beams, Rods, Springs, etc) is even more problematic than it is for shells. For many of these elements an axial stress could be reported. But, for beam elements that stress could not include the effects of beam bending unless details of the beam cross section were available. For some elements (such as a spring) no concept of stress is even correct. As a consequence, we do not report stresses for line type elements. Some recovery may be obtained using the element force output (see section 2.8.20).

5 Troubleshooting

A variety of issues can cause an analysis to fail. Clearly, there are still bugs in the **Salinas** software, and these will continue to be found. However, most problems are identified with problems in the model or other input to the software. This section may help to identify these issues with the goal of completing the analysis properly. Typically the fastest resolution to a problem is to try to eliminate the modeling issues, and only then treat the problem as a potential bug.

Users can troubleshoot **Salinas** issues through stand-alone tools or using **Salinas** capabilities. The following sections will describe some of the ways this may be done. The first part describes the stand-alone tools . The second part describes the ways of using **Salinas** capabilities to troubleshoot problems or issues.

5.1 Stand-Alone Tools

Currently, two tools exist which can help the user debug their mesh file, i.e., **Exodus** file: **Grope** and **Verde**.

5.1.1 Grope

Grope is an ACCESS/SEACAS utility that can be used to interrogate the **Exodus** file. One of the commands in **Grope** that can be used is **check**. It is used as follows:

```
prompt> grope cube.exo
.
.
GROPE> check

 Database check is completed

GROPE>
```

If there are any warning or errors, they will appear before the **Database check is completed** message.

5.1.2 Verde

The Cubit team has developed a GUI-based tool named **Verde**. **Verde** can be used to look at various mesh quality parameters of the **Exodus** file. For questions about **Verde**, please contact the Cubit team at **cubit-dev@sandia.gov**.

5.2 Using Salinas To Troubleshoot

When running **Salinas** on a parallel platform, new users will most likely face issues they are not accustomed to. One of the issues will be in choosing the correct **FETI** section parameters. Section 5.3 can help the user identify and troubleshoot the **FETI** parameters.

The user has to take additional steps before executing the parallel version of **Salinas**. One of the steps is to run **nem_slice** or **yada** to decompose the finite element model. This will produce a load balancing file with a "`.nem`" extension. Using the **Exodus** file and the load balancing file, the next step is to run **nem_spread** to create the partitioned files on the parallel platform where **Salinas** will be executed. Finally, the commands needed to run **Salinas** on the parallel platform need to be learned so that execution of **Salinas** can begin. Many of these steps can cause frustration to the user, but problems with any of these steps are often easily addressed.

The **FETI** solver is one of the most advanced solvers in the world, but it also is sensitive to the decompositions created. Therefore, a model that might appear to be working in serial can fail in parallel due to decomposition issues. Sometimes, the problem can be the model itself, e.g., a model that hasn't been properly equivalenced.

Salinas developers have added various capabilities into **Salinas** to help troubleshoot various issues.

5.2.1 Using The Node_List_File For Debugging Subdomains With ZEMs

The *node_list_file* option is very useful in debugging subdomains that have ZEMs (or RBMs)[§]. To use this feature for debugging,

1. Make sure **prt_debug 3** is set in the **FETI** section. This will produce a `corner.data` file.

2. The "corner.data" file has the following format:

   ```
   NumCorners
   global_id local_id subdomain_id x_coord y_coord z_coord
   .
   .
   ```

3. Use *awk* (or similar utility) to obtain the local_ids of the subdomain from corner.data

[§]Zero Energy or Rigid Body modes

4. Make sure to add an offset of 1 to the local_ids. Put these ids in a file, e.g., sub.corners.

5. Change the **Boundary** section of the Salinas input file to include *node_list_file*:

```
Boundary
  node_list_file="sub.corners"
    fixed
End
```

6. Change the **geometry_file** to point to the subdomain being investigated.

7. Run serial Salinas using the parallel input file

This will help in debugging subdomains that are problematic.

5.2.2 Identifying Problematic Subdomains

Sometimes it is very difficult to identify subdomains that might be problematic. When running an eigen solution (in parallel), a shift is usually specified. Though the shift helps obtain solutions when global rigid body modes exist, this shift can also hide problematic subdomains. This issue also arises when running transient analysis. If a problematic subdomain is suspected, try an eigen analysis with a shift of zero. This will help identify subdomains with ZEMs. If ZEMs are discovered, then section 5.2.1 can help evaluate the source of the ZEMs on that subdomain using a serial version of Salinas.

Sometimes bad subdomains can exist if the global model is not well connected. It is possible to use **yada** to try and create a one processor decomposition of the global Exodus file. If **yada** finds what appears to be disconnected pieces, it will add one processor for each disconnect piece. Once execution is complete, the **color_domains** utility can be used to create an Exodus file for visualization that will have the processor id as an element variable. Or, simply run **nem_spread** on the new decomposition and visualize each subdomain individually.

5.2.3 Problematic Elements and Connectivity

Many problems are caused by "bad" elements. Following are a few issues that come up periodically.

Rotational Invariance can be lost for certain elements such as springs if they are not of zero length. The spring shown in Figure 20 is invariant to rotation about the x axis, but not invariant to rotation about y or z. If we consider an

undeformed rotation about the center of the beam along the z axis we would find that $u_y(1) < 0$ and $u_y(2) = -u_y(1)$. If the spring has **KY**$\neq 0$, then this undeformed rotation actually results in strain energy, $E = 2\,K_Y\,u_y^2$. Thus, the rigid body rotation is no longer a zero energy mode.

This is important for a variety of line type elements including spring, joint2g and gap elements.

Figure 20: Single *Spring* element

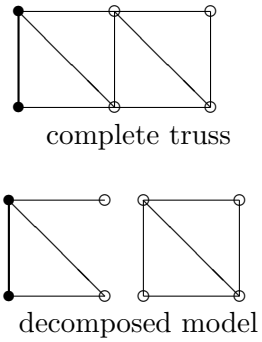
Bad element shape is a major source of problems. For example, we have examined models that have "triangles" where one side is 1/200th the length of the other sides. This produces extremely poor element matrices. In some cases this can destroy the condition of the entire system. Such elements can sometimes be found using the **kdiag** output option described in section 2.8.27.

Decomposition weakness is an issue for trusses (or rods) and some other elements. The truss in the top part of Figure 21 is self sustaining when made of truss elements. However, because truss elements have no rotational stiffness, the decomposed model in the lower part of the figure contains mechanisms. Note that there is no way to decompose the model without introducing such mechanisms.

This does not mean that truss elements must not be used in **Salinas** . There are times when they are the correct element to use. However, extreme care must be taken in their decomposition, and occasionally extra "corner nodes" may be needed to avoid mechanisms (see section 2.4.1).

Poor Connectivity A structure that has poorly connected regions can be very difficult to analyze. If elements have not been properly equivalenced, there can be thousands of zero energy modes in the model. **Salinas** is fairly good at identifying up to a few dozen redundant modes in the best of cases.

Figure 21: Truss Decomposition Issues



5.3 Troubleshooting FETI Issues

5.3.1 Introduction

The Finite Element Tearing and Interconnecting (FETI) solver achieves unprecedented speed and scalability on massively parallel computers. However, it is significantly more complex than a standard direct solver. We discuss a number of the options associated with the solver in the following sections. These options are required to achieve three sometimes-competing goals.

1. Insuring that there is sufficient memory to run on the MP platform.
2. Obtaining the current solution through correct rigid body (or zero energy) identification on the subdomain and on the coarse grid.
3. Tuning the solver to maximize performance.

5.3.2 Standard FETI Block

The default entries for the FETI block are shown below.

```
FETI
    rbm                  geometric
    preconditioner       dirichlet
    corner_algorithm     1
    corner_dimensionality 6
    corner_augmentation  none
    max_iter             200
    orthog               1000
    solver_tol           1e-6
```

```
    grbm_tol              1e-6
    coarse_solver         sparse
    local_solver          sparse
    precondition_solver   sparse
    prt_summary           yes
    prt_rbm               yes
    prt_debug             2
END
```

5.3.3 Memory

The FETI options that directly affect memory usage are listed in the following table. Memory is directly related to the "size" of a subdomain. The number of elements associated with a subdomain can approximately measure the "size". The topology or connectivity of those elements also directly affects the memory since this determines the local sparse matrix structure.

Large memory allocations occur in the following order with the relative importance listed in parentheses. These operations are only done once for linear static/dynamic and eigen analysis in Salinas.

1. Preconditioner (3)
2. Local Solver (2)
3. Coarse Grid (1)
4. Orthog vectors (4)

Preconditioner The lumped preconditioner requires less memory but generally does more iterations than the dirichlet preconditioner which requires more memory. The `precondition_solver` option only affects the memory if the Dirichlet preconditioner is selected. Then the comments in the Local Solver section also apply.

Local Solver The skyline solver typically takes more memory than the sparse solver. For small problems (less than 1000 equations), the skyline solver may require less memory than the sparse solver. Generally the skyline solver is the more robust option particularly when the solution may be singular (i.e. eigenvalue analysis on a floating structure).

Coarse Solver The corner algorithm, corner dimensionality, corner augmentation, and coarse solver options affect the coarse grid memory requirements. The number of equations in the coarse grid can be found in the solution.data file. Reducing the number of equations in the coarse grid reduces the memory required by the coarse grid.

If your model has shell elements, then corner dimensionality 6 results in more memory than corner dimensionality 3. If your model does not have shells, then this option will not affect memory. Corner dimensionality 6 is generally required for good performance on shell models.

Corner algorithm memory requirements are model dependent and are directly related to the interface topology of the decomposed global model. Typically, corner algorithm 0 results in the smallest coarse grids. This is also the least robust corner algorithm. Corner algorithm 3 is the most conservative corner algorithm and typically generates larger coarse grids. It is recommended to start with corner algorithm 1. If problems arise, change to corner algorithm 3.

Both the skyline and sparse coarse grid solvers are redundantly stored on every processor. The same comments about the skyline and sparse solvers found in the Local Solver section apply here too. The parallel sparse (psparse) solver distributes the coarse grid memory among Ns coarse solver processors. Very large coarse grids can be used with this option. If there are any problems found with the parallel sparse solver, please contact me at khpiers@sandia.gov.

Orthogonalization (Ortho) Vectors The number of ortho vectors directly affects the memory requirements of FETI-DP. Generally, you want to select as many ortho vectors as possible given the memory limitations. Ortho vectors decrease the number of iterations required for successive right hand side vectors (eigen/dynamic analysis).

Options that Affect Memory

```
FETI
     preconditioner [lumped/dirichlet]
     precondition_solver [skyline/sparse]
     orthog 200
     local_solver [skyline/sparse]
     coarse_solver [skyline/sparse/psparse]
     corner_dimensionality [3/6]
     corner_algorithm [0,1,2,3,4]
     corner_augmentation [none/subdomain/edge]
END
```

5.3.4 Local Rigid Body Modes

Local rigid body modes (RBMs) refer to the local subdomain stiffness matrix having singularities found during the LDLT factorization and in general the solution will

be corrupted if local RBMs are found. The command "prt_rbm yes" in the FETI block will print the number of local RBMs found for each subdomain in your model. Each subdomain is expected to have zero local RBMs. The following steps can be taken if you find a subdomain with a non-zero number of local RBMs.

1. Reduce the tolerance used in the LDLT factorization, For example, the default value for "rbm_tol_mech" is 1.0E-08, then try "rbm_tol_mech 1.0E-12"

2. If this does not remove the local RBMs, then try changing the corner algorithm while holding the previously set tolerance constant. The recommended and default algorithm is 1. If corner algorithm 1 fails to remove the local RBMs, then try corner algorithm 3.

3. If you have shell elements in the model (and more specifically in the subdomain you have found local RBMs), then "corner dimensionality 6" may be required.

4. For more detailed debugging of RBMs (or ZEMs) for specific subdomains, see section 5.2.1.

5. If you still have local RBMs, contact me at khpiers@sandia.gov and I'll be happy to look at your specific problem.

5.3.5 Global Rigid Body Modes

Global rigid body modes (RBMs) refer to the global stiffness matrix having singularities present. Finding 6 RBMs for a 3D model is expected when performing an eigen analysis with Salinas and the global model does not have any prescribed displacement boundary conditions. FETI-DP can handle this case, but in many cases tolerances have to be adjusted for a particular model.

Finding the incorrect number of RBMs can lead to either stagnation in the FETI solution or the dreaded "relative residual greater than 1" error in Salinas. Troubleshooting this problem can be done in the following fashion.

1. First, determine the expected number of RBMs in your model. Typically in eigen analysis, this is zero (fully constrained), three (2D-floating), or six (3D-floating). The number of RBMs is expected to be zero for transient dynamics.

2. Next, determine how many you are finding with the FETI parameters you have selected. The number of global RBMs are printed to the screen during a Salinas run and printed to the solution.data file. Executing the following UNIX command will find the number of global RBMs found during the last Salinas run. grep "Global RBM" solution.data

3. The parameter "grbm_tol 1.0E-06" will have to be adjusted to find the expected number of RBMs in your model.

4. Decrease grbm_tol if you want to find less global RBMs.

5. Increase grbm_tol if you want to find more global RBMs.

6. For eigen analysis, you may want to use a negative shift (in the Salinas SOLUTION block). Use a shift value equal to the negative of the first anticipated flexible eigenvalue, i.e. $(2\pi f)^2$. This should eliminate all global RBMs, but may slow the solution.

7. If you still have problems with global RBMs, please contact Kendall Pierson at `khpiers@sandia.gov`, and I will be happy to help resolve the problem.

6 Acknowledgments

Salinas is a success based on work by many individuals and teams. These include the following.

1. The ASCI program at the DOE which funded its development.

2. Line managers at Sandia Labs who supported this effort. Special recognition is extended to David Martinez who helped establish the effort.

3. Charbel Farhat and the University of Colorado at Boulder. They have provided incredible support in the area of finite elements, and especially in development of FETI.

4. Carlos Felippa of U. Colorado at Boulder. His consultation has been invaluable, and includes the summer of 2001 where he visited at Sandia and developed the HexShell element for us.

5. Danny Sorensen, Rich Lehoucq and other developers of **ARPACK**, which is used extensively for eigen analysis.

6. Esmond Ng who wrote *sparspak* for us. This sparse solver package is responsible for much of the performance in Salinas and in FETI.

7. The *metis* team at the university of Minnesota. *Metis* is an important part of the graph partitioning schemes used by several of our linear solvers. These are copyright 1997 from the University of Minnesota. Documentation is available at `http://www-users.cs.umn.edu/~karypis/metis/metis/index.html`.

8. Padma Raghaven for development of a parallel direct solver that is a part of the FETI solver.

9. The developers of the *SuperLU* package. This is used in a variety of areas, including a sparse direct complex solver. More information can be obtained at, `http://www.nersc.gov/~xiaoye/SuperLU`.

References

[1] Schoof, L. A. and Yarberry, V. R., "EXODUS II: A Finite Element Data Model," Tech. Rep. SAND92-2137, Sandia National Laboratories, 1994.

[2] Johnson, C. D., Kienholz, D. A., and Rogers, L. C., "FINITE ELEMENT PREDICTION OF DAMPING IN BEAMS WITH CONSTRAINED VISCOELASTIC LAYERS," *AIAA Journal*, **vol. 20**, no. 9, 1982, pp. 1284–1290.

[3] Reese, G., Field, R., and Segalman, D. J., "A Tutorial on Design Analysis Using von Mises Stress in Random Vibration Environments," *Shock and Vibration. Digest*, **vol. 32**, no. 6, 2000.

[4] Farhat, C., Crivelli, and Géradin, M., "Implicit time integration of a class of constrained hybrid formulations - Part I: Spectral stability theory," *International Journal for Numerical Methods in Engineering*, **vol. 41**, 1998, pp. 675–696.

[5] Chung, J. and Hulbert, G., "A Time Integration Algorithm for Structural Dynamics with Improved Numerical Dissipation: The Generalized alpha method," *Journal of Applied Mechanics*, **vol. 60**, pp. 371–375.

[6] Farhat, C. and Roux, F.-X., "A Method of Finite Element Tearing and Interconnecting and Its Parallel Solution Algorithm," *International Journal for Numerical Methods in Engineering*, **vol. 32**, 1991, pp. 1205–1227.

[7] Cook, R. D. and D. S. Malkaus, M. E. P., *Concepts and Applications of Finite Element Analysis*, John Wiley & Sons, third edn., 1989.

[8] Knupp, P. M., "Achieving Finite Element Mesh Quality Via Optimization of the Jacobian Matrix Norm and Associated Quantities : Part II - A Framework for Volume Mesh Optimization and the Condition Number of the Jacobian Matrix," Tech. Rep. SAND99-0709J, Sandia National Laboratories, 1998.

[9] Aklonis, J. L. and MacKnight, W. L., *Introduction to Polymer Viscoelasticity*, Wiley, 1983.

[10] Ferry, J. D., *Viscoelastic Properties of Polymers*, Wiley, 1980.

[11] Hamilton, M. F. and D. T. Blackstock, E., *Nonlinear Acoustics*, Academic Press, 1998.

[12] Carroll, S. K., Drake, R. R., Hensinger, D. H., Luchini, C. B., Petney, S. J. V., Robbins, J. H., Robinson, A. C., Summers, R. M., Voth, T. E., Wong, M. K. W.,

Brunner, T. A., Garasi, C. J., Haill, T. A., and Mehlhorn, T. A., "ALEGRA: Version 4.6," Tech. Rep. SAND2004-6541, Sandia National Laboratories, 2004.

[13] Taylor, R. L., Beresford, P. J., and Wilson, E. L., "A Non-conforming Element for Stress Analysis," *International Journal for Numerical Methods in Engineering*, **vol. 10**, 1976, pp. 1211–1219.

[14] Ibrahimbegovic, A. and Wilson, E. L., "A Modified Method of Incompatible Modes," *Communications in Applied Numerical Methods*, **vol. 7**, 1991, pp. 187–194.

[15] MacNeal, R. H., *Finite Elements: Their Design and Performance*, Marcel Dekker, 1994.

[16] Allman, D. J., "A Compatible Triangular Element Including Vertex Rotations for Plane Elasticity Problems," *Computers and Structures*, **vol. 19**, no. 1-2, 1996, pp. 1–8.

[17] Batoz, J.-L., Bathe, K.-J., and Ho, L.-W., "A Study of Three-Node Triangular Plate Bending Elements," *International Journal for Numerical Methods in Engineering*, **vol. 15**, 1980, pp. 1771–1812.

[18] Felippa, C. A., "The SS8 Solid-Shell Element: Formulation and a Mathematica Implementation," Tech. Rep. CU-CAS-02-03, Univ. Colo. at Boulder, 2002.

[19] Smallwood, D. O., Gregory, D. L., and Coleman, R. G., "A three parameter constitutive model for a joint which exhibits a power law relationship between energy loss and relative displacement," in *Shock and Vibration Symposium*, Destin, FL, 2001.

[20] Segalman, D. J., "An Initial Overview of Iwan Modeling for Mechanical Joints," Tech. Rep. SAND2001-0811, Sandia National Laboratories, 2001.

[21] Segalman, D. J. and Starr, M. J., "Relationships Among Certain Joint Constitutuve Models," Tech. Rep. SAND2004-4321, Sandia National Laboratories, 2004.

[22] Smallwood, D. O., Gregory, D. L., and Coleman, R. G., "A three parameter constitutive model for a joint which exhibits a power law relationship between energy loss and relative displacement," Tech. Rep. SAND2001-1758C, Sandia National Laboratories, November 2001.

A Salinas Example Input Files

The following sections give examples of **Salinas** input files. Note, case sensitivity of the keywords is ignored unless in quotes. The exception is the **#include** command, where the filename following the command must not be in quotes, but case sensitivity is preserved.

A.1 An Eigenanalysis Input File

The following input file will output the first four mode shapes to an **Exodus** output file name *hexplate-out.exo*. A results file, *hexplate.rslt*, will not be created since no results have been selected for output in the **ECHO** section.

```
SOLUTION
         eigen
         nmodes 4
         title 'Obtain First Four Mode Shapes For Hexplate'
END

// The f.e.m. is in hexplate.exo
FILE
         geometry_file         'hexplate.exo'
END

BOUNDARY
         nodeset 77
                  fixed
END

LOADS  // loads are unnecessary for eigenanalysis
END

// Only deformations will be output
OUTPUTS
//         maa
//         kaa
//         faa
         deform
//         stress
//         strain
```

```
END

// No results are output to the text log file, *.rslt
ECHO
//  MATERIALS
//  ELEMENTS
//  JACOBIAN
//  ALL_JACOBIANS
//  TIMING
//  MESH
//  mass
//  INPUT
//  NODES
//  FETI_INPUT
//  DISP
//  STRAIN
//  STRESS
//  MFILE
 none
END


// the following element block is hex.
// exodus tells us it is an 8-node hex.
// The default hex is an underintegraged hex.
BLOCK 44
         material 3
         hex8
END

MATERIAL 3
         name "steel"
         E 30e6
         nu .3
         density 0.288
END
```

A.2 An Anisotropic Material Input File

The following input file is an example of a hexahedral mesh with anisotropic properties.

```
SOLUTION
         eigen
         title 'Example of anisotropic format'
END

FILE
         geometry_file         'anisogump.exo'
END

boundary
        nodeset 4 y = 0
        nodeset 5 x = 0
        nodeset 6 z = 0
end

loads
         // sum of forces on surface should be equal to area
         // imposed forces are additive
         nodeset 1 force = 0.0  0.083333  0.0
         nodeset 2 force = 0.0 -0.041666  0.0
         nodeset 3 force = 0.0 -0.020833  0.0
end


OUTPUTS
//         maa
//         kaa
//         faa
           deform
//         stress
//         strain
END

ECHO
```

```
//  MATERIALS
//  ELEMENTS
//  JACOBIAN
//  ALL_JACOBIANS
//  TIMING
//  MESH
//  mass
//  INPUT
//  NODES
//  FETI_INPUT
//  DISP
//  STRAIN
//  STRESS
//  MFILE
none
END


// the following element block  is all hex
BLOCK 1
         hex8
         material 1
END

MATERIAL 1
         name "anisotropic gump"
         anisotropic
         Cij
         1.346     0.5769    0.5769  0       0       0
                   1.346     0.5769  0       0       0
                             1.346   0       0       0
                                     0.3846  0       0
                                             0.3846  0
                                                     0.3846
         density 1
END
```

A.3 A Multi-material Input File

The next example shows the input for an **Exodus** model with many element blocks and materials. Keyword **lumped** in the **SOLUTION** section causes **Salinas** to use a lumped mass matrix instead of a consistent mass matrix.

```
SOLUTION
         eigen
         nmodes 1
         titile 'Multiple block, multiple material example'
         lumped
END

FILE
         geometry_file         'multi.exo'
END

BOUNDARY
    nodeset 1
    fixed
    nodeset 3
    x = 0
    y = 0
    z = 0
    RotY = 0
    RotZ = 0
END

OUTPUTS    // output only displacements to exodus file
     deform
END

ECHO
    none
END

// element block specifications. One such definition per element
// block in the exodus (genesis) database.
BLOCK 1
```

```
        material 2
        Beam2
END

BLOCK 101
        integration full
        wedge6
        MATERIAL 1
END

BLOCK 2
        material 2
END

BLOCK 102
        integration full
        wedge6
        MATERIAL 2
END

BLOCK 3
        material 3
END

BLOCK 103
        integration full
        wedge6
        MATERIAL 3
END

BLOCK 4
        material 4
END

BLOCK 104
        integration full
        wedge6
        MATERIAL 4
END
```

```
BLOCK 5
         material 5
END

BLOCK 105
         wedge6
         integration full
         MATERIAL 5
END

BLOCK 6
         material 6
END

BLOCK 106
         wedge6
         integration full
         MATERIAL 6
END

// material specifications. Extra materials are acceptable, but
// every material referenced in a necessary "Block" definition,
// must be included here.
MATERIAL 1
         name "Phenolic"
         E 10.5E5
         nu .3
         density 129.5e-6
END

Material 2
         name 'Aluminum'
         E 10.0E6
         nu 0.33
         density 253.82e-6
END


Material 3
         name 'foam'
```

```
        E 100.
        nu 0.3
        density 18.13e-6
END

Material 4
        name 'HE'
        E 5E5
        nu 0.45
        density 129.5e-6
END

material 5
        name 'Uranium'
        E 30e6
        nu 0.3
        density 1768.97e-6
end

material 6
        name 'wood'
        E 200.e3
        nu .3
        density 77.7e-6
end
```

A.4 A Modaltransient Input File

The next example shows the input for a **modaltransient** analysis. Accelerations are output to an **Exodus** file *bar-out.exo*. This example has damping, polynomial and linear functions. Also, sensitivities are calculated.

```
SOLUTION
  modaltransient
    nmodes 10
    time_step .000005
    nsteps 100
    nskip 1
    title 'Test modal transient on prismatic bar'
END

FILE
  geometry_file 'bar.exo'
END

ECHO
//  acceleration
END

OUTPUTS
  acceleration
END

BOUNDARY
  nodeset 1
    fixed
END

DAMPING
  gamma 0.001
END

BLOCK 1
  material 1
END
```

```
MATERIAL 1
  name "aluminum"
  E 10e6
  nu .33
  density 2.59e-4
END

LOADS
  nodeset 3
    force = 1. 1. 1.
    function = 3
END

FUNCTION 1
  type LINEAR
  name "test_func1"
  data 0.0 0.0
  data 0.0150 0.0
  data 0.0152 1.0
  data 0.030 0.0
END

FUNCTION 3
  type LINEAR
  name "white noise"
  data 0.0 1.0
  data 0.0001 1.0
  data 0.0001 0.0
  data 1.0 0.0
END

SENSITIVITY
  vectors all
END
```

A.5 A Modalfrf Input File

The next example shows the input for a **modalfrf** analysis. Accelerations are output to an **Exodus** file *bar-out.frq*.

```
SOLUTION
  modalfrf
    nmodes 10
    title 'Test modalfrf on prismatic bar'
END

FILE
  geometry_file 'bar.exo'
END

frequency
   freq_min 0
   freq_step=10
   freq_max=3000
   nodeset 3
   disp
END

ECHO
//  acceleration
END

OUTPUTS
  acceleration
END

BOUNDARY
  nodeset 1
    fixed
END

DAMPING
  gamma 0.001
END
```

```
BLOCK 1
  material 1
END

MATERIAL 1
  name "aluminum"
  E 10e6
  nu .33
  density 2.59e-4
END

LOADS
  nodeset 3
    force = 1. 1. 1.
    function = 3
END

FUNCTION 2
// this is a smooth pulse with time duration .05
//  it peaks at approximately t=.02 sec with a
//  value of 0.945
  type POLYNOMIAL
  name "poly_fun"
  data 0. 0.
  data 2.0 -8.0e2
  data 0.5 8.9443
END

FUNCTION 3
  type LINEAR
  name "white noise"
  data 0.0 1.0
  data 10000. 1.0
END

SENSITIVITY
  vectors all
END
```

A.6 A Directfrf Input File

The next example shows the input for a **directfrf** analysis. Displacements are output to an **Exodus** file *bar-out.frq*.

```
SOLUTION
directfrf
END

Frequency
  freq_min = 1000.0
  freq_step = 7000
  freq_max = 5.0e4
  disp
  block 1
End

FILE
  geometry_file 'bar.exo'
END

OUTPUTS
disp
END

ECHO
//
none
END

BOUNDARY
  nodeset 1
    fixed
END

BLOCK 1
  material 1
END
```

```
MATERIAL 1
  name "aluminum"
  G 0.8E+9
  K 4.8E+9
  density 2.59e-4
END

LOADS
  sideset 1
  pressure = -1.0
  function=3
END

FUNCTION 3
   type LINEAR
   name "white noise"
   data 0.0 1.0
   data 10000. 1.0
END
```

A.7 A Statics Input File

The following example is a **statics** analysis which will output stresses to the **Exodus** output file *quadt-out.exo*.

```
SOLUTION
        statics
        title '10x1 beam of quadt'
END

FILE
         geometry_file         'quadt.exo'
END

BOUNDARY
           nodeset 1
            fixed
END

LOADS
           nodeset 2
            force = 1000.0 1000.0 0.0
END

OUTPUTS
      stress
END

ECHO
  none
END

// the following element block is quadt
BLOCK 1
         material 1
         QuadT
END

MATERIAL 1
         name "steel"
```

```
            E 30.0e6
            nu 0.25e0
            density 0.7324e-3
      END
```

B Running Salinas on serial UNIX platforms

On serial unix platforms, **Salinas** is run with a single argument, the ASCII input file.

```
salinas example.inp
```

The log file will be written to *example.rslt* if outputs have been specified in the ECHO section. If outputs have been specified in the OUTPUTS section, a new exodus file will be generated. The file name is derived from the `geometry_file` specified in the ASCII input (see section 2.11).

C Running Salinas in Parallel

This section provides an example of how to perform an analysis on the Intel Teraflop (**janus**) using **Salinas**. This implies that the execution of **Salinas** will be in parallel. There is some overhead to running in parallel versus serial. Assuming a **Salinas** text input file exists and an **Exodus** file exists which contains the finite element model, the following steps are needed.

1. Decide on how many processors, *nproc*, are needed.
2. Create an input file for **nem_slice**. The partition software can be executed on a workstation to create a load balance file. The name of this file is specified in the input file for *nem_slice*, and usually has a *.nem* extension.
3. Create your workspace on **janus** on /scratch/tmp_?? - where ?? is (currently) your choice of 1 thru 10.
4. Move the **Salinas** input file, **Exodus** file, and load balance file to your work space on **janus**.
5. Create an input file for **nem_spread**. Execution of **nem_spread** (on **janus**) with this input will create *nproc* **Exodus** files from the master **Exodus** file and move them to the locations specified in the **nem_spread** input file.
6. Modify the **FILE** section of the **Salinas** input file to agree with the number of RAID disks available and the location of the subdomain **Exodus** files created by **nem_spread**.¶
7. Modify the **ECHO** section in the **Salinas** input file using the keyword **subdomain** to indicate which processors should produce text results files. Having all processors output text results files is very slow for large models. By default only the first subdomain will write an echo file.
8. Use the **yod** command to run **Salinas** in parallel.
9. Create an input file for **nem_join** to join your results back into one **Exodus** output file.

Each step is detailed in the following paragraphs. Additional information on parallel execution can be found at *http://jal.sandia.gov* under the **SEACAS** documentation link.

¶RAID - *Redundant Array of Inexpensive (or Independent) Disks.* These are very important to the performance of a parallel computer. Most are no longer independently addressable so the numraid should be 1.

C.1 Number of Processors Needed

Running **Salinas** in parallel requires the user to specify how many processors at a minimum are needed in order to "fit" the problem into available memory on **janus**. First, determine approximately how many degrees of freedom (d.o.f.) are in the model. Then, table 36 can be used to determine the number of processors needed.

Platform Name	Num Procs Needed
ASCI White	$dofs/30,000$
ASCI Q	$dofs/30,000$
ICC	$dofs/30,000$
ASCI Red	$dofs/10,000$
Cplant	$dofs/30,000$
Rogue	$dofs/30,000$

Table 36: Determining Number Of Processors Needed

C.2 Use nem_slice (or yada) to load balance the model

An example of a **nem_slice** input file is, e.g. *junk_slice.inp*,

```
Graph Type = elemental
Decomposition Method = multikl,cnctd_dom
Input ExodusII File = junk.exo
Output NemesisI File = junk.nem
#Solver Specifications
Machine Description = mesh=500
Misc Options = face_adj
#Weighting Specifications
```

This input file will create a load balance file, *junk.nem*, for running **Salinas** on 500 processors. Note, the *face_adj* option is useful for 3-d models to prevent mechanisms from appearing in the decomposed subdomains and is highly recommended for optimal performance.

To create the load balance file, *junk.nem*, simply type

```
workstation_prompt> nem_slice -a junk_slice.inp
```

Or, if using **yada** to create the load balance file,

```
workstation_prompt> yada junk.exo 500
```

The load balancing software, **nem_slice** or bf yada, is typically executed on a serial machine such as a workstation. More detailed information on **nem_slice** is available at *http://jal.sandia.gov* under the link to the **SEACAS** documentation, and for locating **yada** on a particular platforms, see section D.2.

C.3 Janus Work Space

To run **Salinas** in parallel, work space on **janus** is needed. On the /scratch space on **janus**, there are 10 temp directories. Simply choose one, and make a directory using your username, as follows.

```
janus> cd /scratch/tmp_1
janus> mkdir $USER
```

After the work space on janus is set up, move the **Salinas** input file, **Exodus** file, and load balance file (*junk.nem*) to it.

C.4 Using Nem_spread

The load balanced **Exodus** database must be "spread" to *nproc* mini-databases. Each processor reads from its own data file. An example **nem_spread** input file is, e.g. *junk_spread.inp*.

```
Input FEM file                  = junk.exo
LB file                         = junk.nem
Debug                           = 4
------------------------------------------------------------
               Parallel I/O section
------------------------------------------------------------
Parallel Disk Info      = number=18
Parallel file location  = root=/pfs_grande/tmp_, subdir=username
```

Here, `username` must be replaced by the name of the user.

The **Exodus** file and the load balance file need to be defined in the **nem_spread** input file. There are 18 RAID disks currently available on **janus**. These are the number of disks available to which input/output can be performed in parallel. The

FILE section in the **Salinas** input file needs to have the number of raids defined using the keyword **numraid**. Therefore, for janus, **numraid 18**, must appear in the **Salinas** input file. This number must match the parallel disk info line in the **nem_spread** input file.

If running for the first time on janus, proper directories must be established on the RAID disks. Currently, the raids are setup at */pfs_grande/tmp_??* where ?? is a number between 1 and 18 (18 raids). A few `csh` shell commands can make the required directories.

```
janus> foreach i (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18)
foreach? mkdir /pfs_grande/tmp_$i/$USER
foreach? end
janus>
```

To execute **nem_spread**,

```
janus> /cougar/bin/yod -sz 4 nem_spread junk_spread.inp
```

For information on workspace locations for various platforms, see section D.4.

This execution of **nem_spread** will spread *nproc* **Exodus** files onto the RAID disks specified in the input file for **nem_spread**. This location must also be specified in the **FILE** section of the **Salinas** input file as follows, assuming your load balance file is *junk.nem* created for 500 processors,

```
FILE
   geometry_file '/pfs_grande/tmp_%d/username/junk.par.500.%.3d'
   numraid 18
END
```

The "`%d`" after *tmp_* is used in **Salinas** in conjunction with the number of RAIDs available. The "`%.3d`" at the end of the line for the **geometry_file** is used in conjunction with how many processors the load balance file was created with. The following table shows what must be used after *junk.par.*`nproc` for various processors requested.

Condition	Use this
nproc<10	"%.1d"
10≤*nproc*<100	"%.2d"
100≤*nproc*<1000	"%.3d"
1000≤*nproc*<10000	"%.4d"

Since **nem_spread** is a parallel code, **yod** must be used to execute it, using the -sz option to specify how many processors are needed. This number need not agree with the number of processors for execution of the analysis. Typically no more than 20 processors would be used to spread files. The **showmesh** utility can be used to indicate the number of interactive processors available.

C.5 Salinas FILE Section

If a load balance file *junk.nem* is created for execution of **Salinas** for 500 processors, and the number of raids is 18, then the **FILE** section of the **Salinas** input file must look like the following.

```
FILE
  numraid 18
  geometry_file '/pfs_grande/tmp_%d/username/junk.par.500.%.3d'
END
```

C.6 Running Salinas

Once the necessary setup has been done, and a parallel **Salinas** code exists in your work space, then

```
janus> cd /scratch/tmp_1/$USER
janus> yod -sz 500 salinas junk.inp
```

This will run **Salinas** in parallel on 500 processors using the input file *junk.inp*.[‖]

In practice, only a small number of processors are available interactively on **janus**or any other parallel platform. To use a larger number of processors, the NQS queuing system must be used. Help is available under the `man` pages on **janus** under the topics `qsub` and `qstat`. To submit an NQS submission, create a small shell script, such as the following.

```
janus> cat run_it
#!/bin/sh
date
cd /scratch/tmp_1/$USER
yod -sz 500 salinas junk.inp
date
```

[‖]**yod** is the command used on this platform to start a parallel run. On other platforms it may be **mpirun**(most Linux), **poe**(IBM) or some other command.

The NQS job is submitted using `qsub` with a command such as the following.

```
/usr/bin/qsub -lT 90:00 -lP 500 -q snl.day -me run_it
```

This command submits a 90 minute run using 500 processors to the queue `snl.day`. A message will be mailed to you when the run has completed, and output from standard out and standard error will be found in files in your working directory. Status of your run can be obtained using `qstat`. Status of all NQS submissions is available with `qstat -a` or `qstat -av`. Contact *janus-help@sandia.gov* for information on queueing policies and options.

For more information on submitting jobs on various platforms, see section D.5.

C.7 Using Nem_join

Once the analysis run has been completed, the output exodus files will need to be recombined into a single file for visualization and processing. **Nem_join** accomplishes this process. A **Nem_join** input file is very similar to the **nem_spread** input file. An example input file is, e.g. *junk_join.inp*.

```
Input FEM file                  = junk.exo
Scalar Results FEM file         = junk-out.exo
Use Scalar Mesh File            = yes
Parallel Results file base name  = junk-out.par
Number of processors            = 500
Debug                           = 4
------------------------------------------------------------
               Parallel I/O section
------------------------------------------------------------
Parallel Disk Info      = number=18
Parallel file location  = root=/pfs_grande/tmp_,subdir=username
```

To run **nem_join**, simply do the following:

```
janus> yod -sz 4 nem_join junk_join.inp
```

This will create a file *junk-out.exo* in your current directory by combining all the **Exodus** output files located on the RAID disks. This is a standard **Exodus** file which may be visualized and processed using serial tools.

D Execution of Salinas on Various Platforms

D.1 Logging On

Please consult table 37.

ASCI White (SCN)	From wizard LAN:	ssh edison
	From edison:	ssh asciwhite.scf.cln
ASCI Q (SCN)	From wizard LAN:	ssh edison
	From edison:	ssh qfe1.lanl.gov
	From qfe1:	llogin
ICC (SCN)	From edison:	ssh freedom
ASCI Red (SCN)	From wizard LAN:	ssh janus-s
	From edison:	ssh janus-s
Cplant (SCN)	From wizard LAN:	ssh ronne
	From edison:	ssh ronne
ASCI Red (SRN)	From SCICO LAN:	ssh janus
ICC (SRN)	From SCICO LAN:	ssh liberty
Rogue (SRN)	From SCICO LAN:	ssh rogue
CPLANT (SRN)	From SCICO LAN:	ssh ross

Table 37: How To Log On To Various Platforms

D.2 Location Of Salinas Files

Please visit http://jal.sandia.gov/Salinas and click on the **platforms** link on the left. Then click on the link about locations of Salinas files.

D.3 Location Of SEACAS/ACCESS Files

Please visit http://jal.sandia.gov/SEACAS/SEACAS.html for locations of files like nem_spread, nem_join, and nem_slice.

D.4 Workspace Area

Please consult table 38.

ASCI White	/p/gw1/USERID /p/gw2/USERID
ASCI Q	/scratch1/USERID /scratch2/USERID
ICC (SCN)	Input files in: /home/USERID Par file in: /scratch[1-3]/USERID
ASCI Red (SCN)	Input files in: /scratch/tmp_[1-10]/USERID Par files to: /pfs_grande/tmp_[1-18]/USERID
CPLANT (SCN)	/enfs/tmp/USERID
ASCI Red (SRN)	Input files in: /scratch/tmp_[1-10]/USERID Par files in: /pfs_grande/tmp_[1-18]/USERID
ICC (SRN)	Input files in: /home/USERID Par file in: /scratch[1-2]/USERID
Rogue	/u1/USERID (not /u0 home area)
CPLANT (SRN)	/enfs/tmp/USERID

Table 38: Where To Put Files On Various Platforms

D.5 Submitting A Job

Please consult table 39 on how to submit jobs on various platforms. Note that the submission for CPLANT and ASCI Red requires *queue* names. For most jobs using *snl* queue on ASCI Red and using *default* on Cplant will work. However, if problems arise using these queue names, please visit http://jal.sandia.gov/Salinas and click on **platforms** and then click on **links to general information for running on a variety of platforms** for further information.

ASCI White	/usr/local/bin/psub *script*
ASCI Q	/lsf/bin/bsub *script*
ICC (SCN)	/apps/openpbs/bin/qsub *script*
ASCI Red (SCN)	/usr/bin/qsub -lT 3:00:00 -lP 400 -q *queue script*
Cplant (SCN)	/bin/qsub -l walltime=23:00:00,size=120 -q *queue script*
ASCI Red (SRN)	/usr/bin/qsub -lT 3:00:00 -lP 400 -q *queue script*
ICC (SRN)	/apps/openpbs/bin/qsub *script*
Rogue	/usr/local/pbs/i686/bin/qsub *script*
Cplant (SRN)	/bin/qsub -l walltime=23:00:00,size=120 -q *queue script*

Table 39: How To Submit Jobs On Various Platforms

D.6 Checking Job Status

Please consult table 40 on how to check the status of jobs on various platforms.

ASCI White	/usr/local/bin/pstat /usr/local/bin/spjstat /usr/local/bin/spj
ASCI Q	/lsf/bin/bjobs /lsf/bin/bqueues
ICC (SCN)	/apps/openpbs/bin/qstat /apps/maui/bin/showbf
ASCI Red (SCN)	/usr/bin/qstat /cougar/bin/showmesh
CPLANT (SCN)	/bin/qstat /cplant/bin/showmesh /cplant/sbin/pingd
ASCI Red (SRN)	/usr/bin/qstat /cougar/bin/showmesh
ICC (SRN)	/apps/openpbs/bin/qstat /apps/maui/bin/showbf
Rogue	/usr/local/pbs/i686/bin/qstat /opt/xcat/bin/pbstop /usr/local/maui/i686/bin/showbf
CPLANT (SRN)	/bin/qstat /cplant/bin/showmesh /cplant/sbin/pingd

Table 40: How To Check Job Status On Various Platforms

D.7 Sample Scripts

This section gives some example scripts for various platforms. This is only intended to help a user get started quickly. Please visit http://jal.sandia.gov/Salinas and click on **platforms** on the left side of the page, then click on **links to general information for running on a variety of platforms** for further information.

D.7.1 ASCI White

The following script is a sample script used on ASCI White.

```
#!/bin/csh
#PSUB -ln 7
#PSUB -lg 99
#PSUB -c pbatch
#PSUB -tM 1000
#PSUB -lc 10
#PSUB -c 16000Mb
setenv MP_COREFILE_FORMAT STDERR
setenv MP_CSS_INTERRUPT yes
setenv MP_USE_FLOW_CONTROL yes
setenv MP_SHARED_MEMORY yes
setenv MP_THREAD_STACKSIZE 10M
unsetenv MP_INFOLEVEL

cd /p/gw1/USERID/wherever
/usr/bin/poe ./salinas hexplate.inp
```

The **#PSUB -ln 7** requests 7 nodes. On ASCI White there are 16 processors per node, but every processor on each node does not have to be used. In fact, it is recommended that only 15 processors per node be used. Therefore, only $15 \times 7 = 105$ processors should be requested if asking for 7 nodes. The **#PSUB -lg 99** requests that only 99 processors be used on these 7 nodes. The remainder of the script should be left the same except for the **cd /p/gw1/USERID/wherever** line and the **poe ./salinas hexplate.inp** line. This information should be changed based on where the workspace area is and what the input file name for the analysis is.

D.7.2 ASCI Q

The following script is a sample script used on ASCI Q.

```
#!/bin/tcsh -f
#BSUB -o /scratch1/USERID/junk.out
#BSUB -e /scratch1/USERID/junk.err
#BSUB -n 60
#BSUB -W 1:00
#BSUB -L /bin/tcsh
#BSUB -J test_job
#BSUB -q snlq

source /opt/modules/modules/init/tcsh
module load MPI_64bit_R5
```

```
module load CXX_6.5.2
module load fortan_5.5.0

cd /scratch1/USERID
/usr/bin/prun ./salinas hexplate.inp
```

The **#BSUB -n 60** requests 60 processors. Most of the file is obvious and has similar characteristics to other scripts.

D.7.3 ASCI Red (SCN and SRN)

The following script is a sample script used on ASCI Red.

```
cd /scratch/tmp_3/USERID
/cougar/bin/yod -sz 100 ./salinas hexplate.inp
```

Again, the above script should be changed according to the location of the workspace and the name of the salinas input file.

D.7.4 CPLANT (SCN or SRN)

The following script is a sample script used on CPLANT.

```
cd /enfs/tmp/USERID
/cplant/bin/yod -sz 100 ./salinas hexplate.inp
```

Again, the above script should be changed according to the location of the workspace and the name of the salinas input file.

D.7.5 ICC (SCN or SRN)

The following script is a sample script used on ICC (SRN).

```
#!/bin/tcsh
#PBS -A 23232/01.2.22
#PBS -l nodes=48:ppn=2
#PBS -l walltime=03:00:00
cd /home/USERID/wherever
/apps/mpiexec/bin/mpiexec -np 95 ./salinas hexplate.inp
```

The **#PBS -A 23232/01.2.22** is the line that specifies the project and task that will be billed for using the ICC. The **#PBS -l nodes=48:ppn=2** requests 48 nodes and 2 processors per node for a total of 96 processors. However, the mpiexec only uses 95 of the 96. Finally, the above script should be changed according to the location of the workspace and the name of the salinas input file.

D.7.6 Rogue

The following script is a sample script used on Rogue when using an IP executable.

```
#!/bin/bash
#PBS -l nodes=1:ppn=2,walltime=3:00:00
#PBS -N hexplate
#PBS -q serial
cd /u1/USERID/junk
/usr/local/mpi/sierra/mpich/1.2.5.2/gcc-3.2.2/bin/mpirun -np 2
-machinefile $PBS_NODEFILE  ./salinas  hexplate.inp
```

The following script is a sample script used on Rogue when using a GM executable.

```
#!/bin/bash
#PBS -l nodes=1:ppn=2,walltime=3:00:00
#PBS -N hexplate
#PBS -q scico

cd /u1/mkbhard/hexplate
/usr/local/mpi/sierra/mpich/1.2.5..12-gm-2.0.12/gcc-3.2.2/bin/mpirun.ch_gm
-v --gm-kill 5 PBS_JOBID=$PBS_JOIBID DISPLAY=head01:0 -TMPNAME=/scr/$PBS_JOBID
-np 2 -machinefile $PBS_NODEFILE ./salinas_gm  hexplate.inp
```

Note that in the above two scripts, the user only has to change the number of nodes being requested, the walltime, the name of the salinas executable, the name of the salinas input file, and the number of processors for the -np argument.

D.8 Special Considerations

On CPLANT (ROSS or RONNE), the input files should be located in the parallel file system listed in table 38. However, in order to utilize the parallel file system optimally, the **geometry_file** in the **Salinas** input file needs "enfs:" as a prefix, e.g., a FILE section in a typical **Salinas** input file should look as follows when running on CPLANT:

```
FILE
  geometry_file "enfs:/enfs/tmp/mkbhard/aff/temp%d/aff.128.par.%.3d"
  numraid 1
END
```

Also, sometimes **Salinas** will abort on CPLANT with the message "Too Many Unexpected Messages". This is an MPI implementation issue on CPLANT. One fix is to set the following environment variable: **setenv MPI_UNEX_MAX 2048**, 2048 is the maximum number allowed for this. While this does not always fix the problem, for certain small problems it does the trick.

E CF FETI

We have found it advantageous to maintain a stable, fixed linear solver, while continuing significant significant solver development with Charbel Farhat originally at the University of Colorado at Boulder (or CU), and now at Stanford University. However, in many respects the stable version is some sort of copy of the development code, called "CF" code.

E.1 Features of CF solver

The current CF solver has a number of features that we have not yet merged into the stable version. This may warrant use of this code for some analysis. The primary features are listed below.

Complex Solver The templated nature of this code permits solution of both real and complex systems of equations. This means that the solver may be used for direct frequency response calculations in parallel.

Contact The solver is designed to understand contact. While we have limited experience to report at this time, the solver takes the contact information and computes the response directly. In this sense, it is a nonlinear solver. It is anticipated that this methodology could permit much more efficient calculation of contact response.

Mortars The solver also directly accepts Tied Surface information (see section 2.17). Again, internal to the solver, appropriate mortar elements are constructed, and solution is performed. Use of mortar elements provides a means of consistently computing the response at an interface. Thus, mismatched meshes should still pass the patch test. We also hope to be able to better handle the large numbers of constraints that can be introduced at these surfaces.

E.2 Limitations of the Solver

There are some limitations and restrictions that should be understood in using this solver.

Parameters Some parameters are invalid, and others are added to provide the additional functionality. See the table below.

Robustness

Constraints Constraints are currently not supported in the complex portion of the solver. This will soon change. 1-27-04.

Testing While we have tested the solver on our test suite, there is much less available history at Sandia for the solver. Some testing also occurs at Stanford of course.

The following table lists parameters that are added, deleted or modified with respect to the Sandia FETI-DP solver. For the standard parameters see section 2.4 and table 7.

Table 41: CF FETI Parameter Modifications

Parameter	Change	Description
rbm_method	modify	only `geometry` accepted
outerloop_solver	add	`cg, gmres`
corner_aug_rbm_type	add	`translation, all, none`
corner_algorithm	modify	*integer*
numWaveDirections	add	*integer*
crbm_tol	add	*Real number*
weighting	add	`Topological, Stiffness`
mpc_method	add	`Dual, Primal`
mpc_submethod	add	`None, Full, PerFace, PerSub, Diag, BlockDiag`
mpc_tol	add	*Real number*
pivoting	add	`on, off`
mpc_solver	add	`skyline, sparse, spooles`
mpc_weighting	add	`topological, stiffness`
multibody	add	`0, 1 or 2`

Some of these parameters are described below.

corner_algorithm These are different from the FETI-DP parameters.

1 standard old

3 Three per neighbor

5,6,7,8 use nQ= 1, 2, 3, and 4 respectively, where nQ+2 is # of touching subdomains

numWaveDirections number of wave directions used to construct Q matrix in FETI-DPH. range 0-13, default = 3.

multibody If equal to "0" use externally defined body parameter passed in CF_Feti constructor. For multibody = "1" we force all subdomains to be treated as

a single body. For multibody = "2" then use the FETI_DPC cornerMaker to find bodies. The default is "0".

Regarding multibody problems such as contact and tied surfaces, there are three alternatives for determining which body each subdomain belongs to. The default "multibody 0" is to do the body decomposition on the salinas side and pass the body id to the CF_FETI constructor. This is currently not implemented, you are just setting body = 1 in CF_FetiSolver.C for all cases which obviously won't work for multibody. However, by selecting "multibody 2" you can choose to ignore the salinas body decomposition and let the our CornerMaker algorithm do it for you. Unfortunately this algorithm doesn't always get it right for some odd cases, and although we are working to improve it you will eventually need to implement the body decomposition on your side because our CornerMaker is CMSoft and will have to be replaced with Kendall's CornerMaker which doesn't do this. The third option is "multibody 1" which forces all subdomains to be treated as a single body.

As currently implemented, it is necessary for every body to be totally independent (i.e. not share any nodes or RBMs) with the exception that they can be connected by MPCs. So to solve a problem with a mechanism you need to split the body containing the mechanism into 2 or more bodies (creating duplicate nodes as required) and then re-tie them with MPCs. This could possibly be done as a pre-processing step on the salinas side.

Index